ADSA - A SOFTWARE PACKAGE FOR AUTOMATIC DESIGN OF SYSTOLIC ARRAYS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

SYED AFZAL HUSSAIN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY 1990

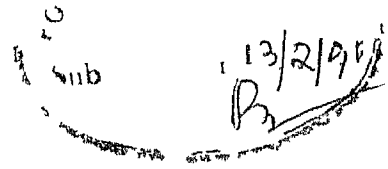
EE-1980-M-HUS-ADSA

Dedicated to my parents

Murthuza Hussain

and

Tahera



CERTIFICATE

This is to certify that the work embodied in this thesis entitled ADSA - A Software Package For Automatic Design of Systolic Arrays has been carried out by Mr SYED AFZAL HUSSAIN under my supervision and the same has not been submitted elsewhere for a degree

(Dr Sumana Gupta)
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

ABSTRACT

This work presents a systematic method for transforming an algorithm represented by mathematical expressions into a systolic array architecture Systolic arrays are highly structured architectures tailored to a specific application They have specific architectural properties such as *simple processing elements (cells) simple and regular data and control communication* and *local cell interconnections*

The method consists of three major steps algorithm representation algorithm model and architecture specification The algorithm representation involves the translation of the algorithm into a set of locally recursive equations In algorithm model step a Dependence Graph (DG) is obtained from these recursive equations From this model the computations are first scheduled and then grouped among a set of cells such that the systolic array characterstics are preserved This grouping of computations is done via geometric projections The valid projection directions, referred to as *projection vectors* are systematically determined from the DG In architecture specification step processor basis A corresponding to each valid projection vector is determined Then a geometric transformation matrix $T = \begin{bmatrix} s^t \\ A^t \end{bmatrix}$ is formed The design information such as number of cells, operations performed in each cell and data timings are extracted from the transformed index set Cell interconnections and data movement are extracted from the

transformed dependencies

The method produces all possible (partitionable/ non-partitionable) systolic solutions for the algorithm under consideration and is the *simplest and computationally less intensive* of all the existing methods

transformed dependencies

The method produces all possible (partitionable/ non-partitionable) systolic solutions for the algorithm under consideration and is the *simplest and computationally less intensive* of all the existing methods

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to *Dr Sumana Gupta* for her ever available counsel and guidance She has been a constant source of inspiration and motivation during all phases of this work

I take this opportunity to thank *Dr Anil Mahanta* for introducing me to this topic and *Dr M M Hasan* for providing me an insight into the VLSI technology Coloquies with *Mr Usman* on the hardware issues of the systolic arrays deserves special mention

The personal communication of *Dr Michael Shanblatt* of Michagan State University and *Dr Patrice Quinton* of IRISA Research centre Rennes Cedex France is deeply appreciated

May Allah reward my brother *Mr Zakir Hussain*, for extending an unstinted financial support throughout my academic career Leisurely chat with *Masood Rais Hari Muqtada Shahab Razi Muzammil Nagaraj Saleem, Sameer Mansoor Paramjit Govinda Rajan, Jaya Raja and Sai* and *Abrar Hussain* has been stimulating

Finally, I would like to thank, *one and all* who helped me in completing successfully the stupendous task of preparing the *thesis*

- HUSSAIN SYED AFZAL

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Will you walk a little faster? said a whiting to a snail

The revolution in VLSI technology allows the implementation of powerful computational systems on a VLSI chip using multiple and regularly connected processing elements These VLSI systems that combine pipelining and multiprocessing have been referred to as systolic arrays [1] Many designs using the systolic array concept exist for computational algorithms in signal and image processing and matrix manipulation [1-5]

The systematic transformation of an algorithm into a systolic array is a design automation problem of current interest The development of a systematic methodology for transforming algorithms into systolic arrays has been addressed [2 5-15] The power as well as the limitations of these methodologies have been recorded [19] In many of the existing methodologies the final systolic array is the result of several ad hoc modificatio ns to the original design The others use linear transformations and often choose the transformations in an ad hoc manner

In general, the systematic construction of systolic arrays can be grouped into three major steps [19]

1 Algorithm Representation An algorithm expressed by some high-level construct e g mathematical expressions is represented in a suitable format such as recursive equations

2 Algorithm Model The algorithm representation from step 1 is modeled for systematic systolic design

3 Architectural Specification From the algorithm model the systolic array specifications such as the number of cells design dimension, cell interconnections operations performed in each cell data movement and data timing are extracted

The above three steps may or may not all be contained in a given methodology Furthermore each major step may consist of several sub-steps

This research describes a simple methodology for systematic construction of systolic arrays and includes all three major steps For step 1 the algorithm is represented by recursive equations In step 2 these recursive equations are represented by a graph which portrays the dependences among the computations This graph is known as Dependence Graph [DG] From this model, various computations are first scheduled and then grouped among a set of cells (processing elements) such that the systolic array characterstics which includes the use of simple cells, simple and regular data and control communications and local cell interconnections are preserved

This grouping of computations is done via geometric projections Not all geometric projections will produce systolic arrays The valid projection directions referred to as projection vectors are systematically derived from this DG In step 3 processor basis A corresponding to each valid projection vector is obtained Then a geometric transformation matrix $T=\begin{bmatrix} s & t \\ A & t \end{bmatrix}$ is formed The design information such as number of cells, operations performed in each cell and data timing are extracted by applying this transformation matrix to the index set of computations Cell interconnections and data movement are extracted from the transformed dependence vector matrices

In general algorithms suitable for systolic implementation are highly structured It is assumed that the number of different systolic solutions for any given problem is problem size independent Therefore, this systematic systolic array design approach is first applied to the smaller sized problems The design information such as the index transformation matrices and the cell interconnections are produced The design information is then easily extrapolated to larger problem sizes to produce corresponding larger systolic arrays

1 1 PROBLEM STATEMENT

Many approaches exist for constructing systolic arrays from algorithms or from initial non-systolic designs For these methods in general, a systolic solution is the result of both

systematic and ad hoc procedures There are approaches which provide systematic methodology for constructing systolic arrays [9 12] However these approaches may not satisfy the simple cell property of systolic arrays The existing methods do not directly incorporate the properties of systolic arrays in the design process It seems appropriate that knowledge of legal communication topologies (such as local connectivity) modularity (such as regular communication) and operation complexities (such as simple arithmetic operations) should be used in designing such systems with such properties A good approach for constructing systolic arrays should use certain algorithm characterstics to control cell complexity data communication and cell interconnectivity Faroughis approach [13-15] is directed towards incorporating the array characterstics in the design process But this method often overlooks the pipelining mechanism of systolic arrays while decomposing the recurrence equation of the algorithm into simple computations (to determine the algorithmic feature interrelationships) Further all these methods are computationally intensive

For a given algorithm there may exist many different systolic solutions, each with different characteristic parameters such as the number of cells, throughput total execution time, cell interconnection pattern and data movement All the systolic solutions for a given algorithm need to be explored in order to select the best solution for the specific application This

work presents a simplified approach for constructing systolic arrays that incorporates the properties of systolic arrays in the design process and produces all the existing solutions for a given algorithm description

1 2 ORGANIZATION OF THE TECHNICAL REPORT

Chapter 2 presents the background to this research In particular Section 2 1 describes the systolic array architecture and its advantages In Section 2 2 the literature on the systematic construction of systolic arrays is reviewed

Chapter 3 presents the simple methodology developed for constructing systolic arrays In particular, Section 3 1 presents the algorithm representation Section 3 2 presents the algorithm model which involves the construction of DG and obtaining valid schedule and projection vectors from this DG Section 3 3 presents the architecture specification

In Chapter 4, the architectures obtained for various algorithms with the aid of ADSA (Automatic Design of Systolic Arrays) package are presented These algorithms include Upper triangular linear systems of equations UX=B Matrix-Matrix multiplication Discrete Fourier Transform LU decomposition FIR filter, recursive convolution and IIR filter

CHAPTER 2

BACKGROUND

" When we mean to build
we first survey the plot
then draw the model And
when we see the figure of
the house then must we
rate the cost of erection

WILLIAM SHAKESPEARE

The properties and advantages of systolic arrays are presented in Section 2 1 Section 2 2 presents a brief review of the existing methods for constructing systolic arrays either from an algorithm description or from an initial non-systolic design

2 1 SYSTOLIC ARRAYS AND THEIR PROPERTIES

Special purpose computer systems are typically used to meet specific performance requirements or to off-load computations that are too time-consuming for a general purpose computer Since these systems have limited applicability their cost-effectiveness has always been the deciding factor in guiding their development With evolving VLSI circuit technology, the design cost of a special-purpose system can be

held down only if appropriate architectures are used Systolic arrays have been characterized as one such architecture

Systolic array architecture consists of

1 simple processing elements (cells)
2 simple and regular control and data communications
3 local cell interconnections
4 extensive pipelining and multiprocessing [i 18]

These cells in general perform simple operations involving a small number of operands hence providing simple communication among the neighboring cells The complete array is constructed by repeating the cells over the chip area Regular communication implies that the design can be made modular Larger systems can be designed by combining the proven smaller ones The local cell interconnections provide simple and regular wire routing, making the design of these special purpose devices cost effective The use of pipelining and multiprocessing enable one to reach the performance requirement of a special-purpose system In a systolic design data flows in a pipelined fashion between the cells and communication with the outside world is performed only at the boundary cells All or portions of the boundary cells have input/output ports

Kung has defined systolic designs as semi-systolic if global data communication is used and pure-systolic if no global

communication is used [1] In global communication data is broadcasted to or collected from some or all of the cells With no global communication data is passed from cell to cell and each datum is used only in one cell at any given time When the number of cells increases using global communication results in long connection wires expanding these non-local communication paths becomes difficult without slowing down the system clock In general pure-systolic is preferred but the correct choice of semi or pure-systolic design depends on the chip I/O limitations the specific application and the environment in which the chip is used The cells in a systolic design operate synchronously The need to distribute the clock signal to every cell in the design without clock skew is one problem of systolic architecture

Another architecture similar to the systolic architecture is the Wavefront Array Processor [16 17] Unlike the systolic array a wavefront array does not operate synchronously Here the operations of each cell are controlled locally and depend on the availability of the input data and on the delivery of the previous output data to the neighboring cells Due to this local control of operations skewing of the input data generally needed in systolic designs may be avoided in wavefront designs

Although a systolic array is usually a special purpose device customized to a specific algorithm they can be designed with

programmable cell functions and programmable cell interconnection patterns to be used for a small subset of different algorithms [18] In this case different functions may be computed in the same array by programming (reconfiguring) the array The number of different systolic arrays that can be programmed into a reconfigurable array is limited by the number of different cell functions and the number of different cell interconnections Reconfigurable systolic arrays are usually pure-systolic

In general a systolic architecture is used to implement a regular and compute-bound problem A problem is compute-bound if the total number of operations is larger than the total number of inputs and outputs A problem is regular when repetitive computations are performed on a large set of data [1] Since the introduction of the systolic architectures numerous systolic designs for various known problems have been proposed [1-5] Automatic construction of systolic designs has only recently been addressed [2 5-15] The quality of a systolic design is evaluated by its correctness delay data flow timing, number of cells (chip area), modularity and many other factors such as accuracy, flexibility, fault tolerance and cost

2 2 AUTOMATIC SYSTOLIC DESIGN

In this section existing methodologies for designing systolic

arrays are reviewed These methodologies are grouped under seperate titles based on related techniques

2 2 1 Z-OPERATOR APPROACH

The Z-operator which is also referred to as the delay operator is defined as

$$Z^{j} \; x_i = x_{i-j}$$

or

$$Z^{-j} \; x_i = x_{i+j}$$

where the indices indicate displacement in time or space When the Z-operator is applied to a mathematical expression with indexed variables the index differences are indicated by the powers of Z For example applying the Z-operator notation to the Finite Impulse Response (FIR) filter defined by

$$y_i = \sum_j a_j * x_{i-j}$$

yields

$$y_i = \sum_j a_j * Z^{j} \; x_i$$

The new expression is then manipulated symbolically using the mathematical properties of the Z-operator Implementations with different properties can be derived [5] By direct hardware interpretation of the expression correctness of the implementation is assured However in general, the direct hardware interpretation does not have the important

characterstics of the systolic architecture Further ad hoc modifications to the designs are needed

2 2 2 INDEX TRANSFORMATIONS

Cappello and Steiglitz use linear transformations on recurrence equations [5] Starting with recurrence equations describing the algorithm a canonical representation is obtained by adjoining a time index to the definition of the recurrence For example

$$y_{i\ j\ t} \leftarrow y_{i\ j\ t-1} + w_j * x_{i-j}$$

is the canonical representation of the FIR filter The coordinates of the left side of the canonical form indicate the location where the computation on the right takes place This gives a geometric interpretation to the recurrence relations A geometric representation together with the ordering rule, indicates what data moves, where it moves to and when it moves With various linear transformations of the geometric representation the original computational locations change thus producing a different geometric shape Then, by factoring out the time dimension (geometric projection) different designs are produced The transformations are selected in an ad hoc manner and in [5] it was also shown that certain transformations will lead to AP^2 optimal design where A is the area and P represents the period

Another method that uses the index transformation is reported

in [10-12] The organization and operations of a systolic design are described by a set of 3-tuples (G F T) G represents the network geometry the position of each cell identified by its Cartesian coordinates The function F associated with each cell represents the operations performed by that cell T represents the network timing and it indicates for each cell the time when the operations F occur and when the data communication takes place

Given an algorithm in the form of a high-level language, the loop bodies are ordered in a lexicographical ordering to eliminate the broadcasts One or more data-dependent vectors correspond to each variable Let $x(I_1)$ and $x(I_2)$ be two variables generated at a statement S using the indices I_1 and I_2 Variable $x(I_2)$ is said to be dependent on a variable $x(I_1)$ if $I_1 < I_2$, in the lexicographical sense and $x(I_1)$ is an input variable in the statement S Then the vector $d=I_2 - I_1$ is called a data-dependent vector for variable x

Let d_1, d_2 , d_m be the set of data-dependent vectors calculated for the algorithm Let $D= [\, d_1\ d_2 \quad d_m]$ be the data-dependent matrix of the algorithm A linear transformation $T= \begin{bmatrix} \Pi \\ S \end{bmatrix}$ when applied to D, will generate the data-dependent matrix of the new structure that is $TD = D'$ Matrix $D = [d_1 , d_2', \quad d_m]$ represents the modified data dependences in the new index space and is favoured by VLSI Here Π is related to time and S is related to the geometric properties of the

algorithm such as cell interconnections The transformation matrices are determined systematically from a set of predetermined cell interconnections which is represented by P Possible S matrices are computed according to the diaphantine equation S*D=P*K where matrix K indicates the utilization of primitive interconnections in matrix P

This approach does not satisfy a property of systolic arrays namely simple cell structures The operations required in the body of the inner loop determine the cell complexity Those algorithms with complex loop bodies will produce complex cell structures as the data dependence vectors which are used as a set of algorithm characterstics are not exploited to obtain an information about the cell complexity Furthermore this method is computationally more intensive To obtain all possible S s the above equation needs to be solved as many times as there are K matrices and this number is usually very large For example, for matrix multiplication algorithm described by A=B*C where A B and C are square matrices of size (3X3) with $P=\begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 0 & -1 & 1 & -1 & 1 \end{bmatrix}$ there exists 729 possible K matrices of size (9X3)

Li and Wah s approach is to model an algorithm represented as recurrence processes into a non-linear integer programming problem [6] Starting with an algorithm described as a set of linear recurrence equations, this method derives three classes of parameters, velocities of data flows spatial distributions

of data and periods of data The relationships among these parameters are represented as constraint vector equations which must be satisfied in order to obtain a correct design The performance of a design can also be expressed in terms of the defined parameters Performance can be defined as execution time or the product of the square of execution time and the number of processors Optimal designs are then searched in the space of solutions satisfying the constraint equations This search is done by ordered enumeration over a limited search space in time polynomial to the problem size From the definition of the recurrence equations the functional description of the cells is derived From the parameters mentioned above, the interconnections among cells are found

In Faroughi s method [13-15], the algorithm represented by mathematical expressions, is first decomposed into a set of equivalent simple computations which are grouped into subsets based on the required set of operations and same type operands Then features, which include the operands of each simple computations the lexicographical index differences of each pair of operands of a simple computation and the lexicographical index differences of the respective operands of each pair of simple computations, are extracted from the simple computations The properties of systolic arrays are represented in terms of feature interrelationships A set of information referred to as valid projection vectors is systematically determined from the feature interrelationships

But this method often overlooks the pipelining mechanism of systolic arrays while decomposing the recurrence equation of the algorithm into simple computations And this is the reason why this method has failed to obtain a systolic architecture for the solution of lower triangular linear systems of equations Lx=b Furthermore the set of feature interrelationships to find the valid projection vectors is usually very large even for a small sized problem and many of the feature interrelationships are trivial

Miranker and Winkler s method [8] is similar to that described in [10-12] Given an algorithm in a high-level language with nested do-loops, loop index expansions is performed in an ad hoc fashion to eliminate the broadcasts The normal order of the operations corresponds to a set of indices Γ which is referred to as the nodes of the computations Γ is represented by a graph structure For example

B(2 4)=B(1,4)

and

C(2,4)=C(2,3)+A(2,4)*B(2,4)

indicate that these statements require computations at nodes (1 4),(2,3) and (2 4) These nodes are called the domain of dependence for node (2,4) In the structure representing Γ dependence vectors (edges) are formed based at node (2,4) which point back to the nodes in its domain of dependence For this example the dependence vectors are (-1,0), (0 -1) and (0,0) and

are denoted by Γ^*

For systolic design Γ^* is homogeneous i e the dependence status at one node is the same as that at any other node Furthermore two structures are isomorphic as graphs if there is a 1-to-1 correspondence f between them such that if (p-q) is a dependence vector then f(p)-f(q) must also be a dependence vector where p and q are nodes Different structures result by mapping p-q onto f(p)-f(q) These mapping are affine mappings and they are given by the linear maps An mXn integer matrix A is selected as a linear mapping Let d_1 d_2 d_m be the dependence vectors of a structure then $Ad_1 = d_1$ for i=1 m gives the dependence vectors of the new structure Conditions for the existence of A are given in [8] The first element in every d_1 corresponds to time and the remaining elements indicate the space For spatial representation of the systolic design, a universal planar systolic array (in 2-D) is constructed where each connection corresponds to a dependence vector in space A universal planar systolic array contains all of the possible dependence vectors The last n-1 elements of every d_1 represent the dependence vector in space and correspond to a portion of the universal systolic array

2 2 3 DEPENDENCE GRAPH TRANSFORMATION

Quinton s method [9] uses quasi-affine mappings Given a system of n uniform recurrence equations defined over some convex

subset D in Z^M (D can be thought of as the index set of recurrence) and with some characteristic dependence vectors (which together define a dependence graph) this method starts by finding a timing function that maps points of D into time This requires identification of a convex space of feasible timing functions from which one can be chosen heuristically Such space can be found systematically from the knowledge of dependence vectors and D Next an allocation function is chosen and it projects D into space along a preselected direction which is chosen so that two points in D with the same image under the timing function do not map into the same point in space Once the timing and allocation functions are known systolic array can be systematically constructed from D The procedure maps each point of D into a cell which computes the recurrence function and receives (sends) data from (to) the cells which are the image of points dependent (depending) on the point under consideration with delays given by the timing function

2 2 4 RETIMING TRANSFORMATION

Leiserson's approach is not the transformation of an algorithm to a systolic design but rather the application of a retiming transformation to an existing synchronous system (not necessarily systolic) in order to improve the performance This method [7] starts with the design of a synchronous circuit whose correctness is obvious or easily verifiable This

design is modeled as a finite rooted vertex-weighted edge-weighted directed multigraph where nodes represent functional cells and edges represent interconnections Weights represent delays of nodes and register delays of interconnection Retiming and k- slow down transformations are then applied to the original design in order to obtain a systolic design without global broadcasts Optimal retiming transformations can be selected so that the transformed circuit has the smallest clock period by reducing this problem to an efficiently solvable mixed-integer linear programming problem Optimal retiming transformations can also be selected so that the total number of registers is minimized by solving a linear-programming dual of a minimum-cost flow problem Extensions of the optimization procedures used can also take into account fan-out interconnection bus width, multiple hosts host timing constraints and geometric constraints like the number of registers per interconnection

Kung's retiming transformation is applied to the time-invariant Signal Flow Graph (SFG) representation of algorithms [2,18] In the SFG in general a node represents an arithmetic or logic operation performed with zero delay and an edge represents a function or a delay Let D and D represent delay elements and lower case letters represent constants An SFG will have on its edges either a delay element or a constant indicating a multiplication by that constant

There are two classes of SFGs those with local interconnections and those with global interconnections Since the SFG with global interconnections such as the FFT will require spatially global interconnections they are not suitable for systolic designs Here only the SFGs with local interconnections are considered such as the FIR

A general systematic procedure for transforming SFG s into systolic designs is based on two simple rules

Rulei Time-Scaling All delays may be scaled that is D → nD Similarly the corresponding input and output rates have to be scaled with respect to the new time unit D by a factor n

Let a cut-set of a SFG be defined as a minimal set of edges which partitions the SFG into two pats The edges of the cut-set are grouped as inbound and outbound edges depending on the assigned directions to the edges

Rule2 Delay- Transfer Advancing k time units on all the outbound edges and delaying k time units on all the inbound edges and vice versa, will not affect the general system operation The effect of lags and advances cancel each other in the overall timing

Based on these two rules it was shown that any zero-delay edge that is non-temporally local can be transformed into a

nonzero-delay edge This is done by selecting a good cut-set that includes the target edge (the zero-delay edge) but does not include zero-delay edges in the opposite direction to the target edge The cut-set could include zero-delay edges going in the same direction as the target edge According to Rule 2 the nonzero-delay edges in the opposite direction can give one or more spare delays to the target edge(s) If there are no spare delays Rule 1 is applied to increase the number of delays in all the edges It was shown that there exist 'good cut-sets for SFG with local interconnections [2 18]

CHAPTER3

A SIMPLE METHOD FOR DESIGNING SYSTOLIC ARRAYS

When you wish to produce a result by means of an instrument do not allow yourself to complicate it

-LEONARDO DA VINCI

This chapter presents the *simple method* developed for constructing systolic arrays from algorithms In particular section 3 1 presents the *algorithm representation* In section 3 2, *algorithm model* is presented which consists of developing the Dependence Graph (DG) and extracting the valid schedule and projection vectors from the DG Section 3 3 presents the extraction of *architectural details* from the information provided by algorithm model step

3 1 ALGORITHM REPRESENTATION

The compute bound algorithms under consideration are highly structured and can be described in a closed form such as a mathematical expression Consider the algorithm of the *solution of lower triangular linear systems of equations* LX=B where L is an NXN lower triangular matrix and X and B are NX1 vectors

A closed form description of this algorithms is

$$x_i = (b_i - \sum_{j<i} l_{ij} * x_j) / l_{ii} \quad \text{for } i = 1\ 2\ 3 \quad N \qquad (3\text{-}1)$$

To indicate the calcultations more explicitly for N=4 the eq (3-1) can be expressed as follows

$$\begin{aligned} x_1 &= (b_1) / l_{11} \\ x_2 &= (b_2 - l_{21} * x_1) / l_{22} \\ x_3 &= (b_3 - l_{31} * x_1 - l_{32} * x_2) / l_{33} \\ x_4 &= (b_4 - l_{41} * x_1 - l_{42} * x_2 - l_{43} * x_3) / l_{44} \end{aligned} \qquad (3\text{-}2)$$

To achieve the maximum parallelism in an algorithm the data dependencies in the computations must be studied carefully If there are no data dependencies between the operations then they can be executed at the same time in a parallel computer Eq (3-1) indicates that the resultant x_i s are determined from certain specified products which are subtracted from b_i s The order in which the products are computed and subtracted from b_i is not important This provides considerable freedom as to how these calculations are to be performed In general one natural order for the calculations is in the order of data availability In this algorithm, x_1 is calculated first and then used in the calculations of x_2, x_3 and x_4 and this process repeats with all elements of vector X

A concise and convenient expression for the representation of many algorithms is to use recursive equations The derivation of recursive equations is often straight forward The recursive equation for the algorithm under consideration is

$$x_i^j = x_i^{j+1} - l_i^j * x_j \qquad \text{for } j < i$$

$$x_i^0 = x_i^1 / l_i^1 \qquad \text{for } i=1\ 2\ 3 \quad N \qquad (3\text{-}3)$$

where j is the recursion index $j=(N-1) \quad 2\ 1$ and

$x_i^1 = b_i \qquad l_i^j = l_{i\ j}$ and $x_i = x_i^0$

The construction of the DG from this recursive equation is the topic of the next section

3 2 ALGORITHM MODEL

A dependence graph (DG) is a graph that shows the dependence of the computations that occur in an algorithm For the algorithm under consideration x_i^j is directly dependent upon x_i^{j+1} l_i^{i-j} and x_{i-j} By viewing each dependence relation as an arc between the corresponding variables located in the index space a DG as shown in Fig 3-1 will be obtained

The DG has two kinds of cells, a cell performing *multiplication and subtraction* operation (MS cell) and a cell performing *division* operation (D cell) Such a DG is termed as non-homogeneous DG

In Fig 3-1, the value x_{i-j} of each element of vector X should be 'broadcast to all the index points having the same $(i-j)$ index Typically, broadcasts are signaled by missing indices of variables in the loop In general this means that global communication is involved in array processor design Thus such broadcasting of the variables has to be replaced by local communication whenever possible

An algorithm is localized if all the variables are dependent upon the variables of the neighboring nodes only Thus a locally recursive algorithm is an algorithm whose corresponding DG has only local dependencies i e the length of each dependence arc is independent of the problem size [18] In order to avoid broadcast and to introduce pipelining we first compute all missing indices and introduce new artificial variables such that *for each generated variable there is only one destination*

For the algorithm under consideration the following forward recurrence equation has all the variables pipelined and executes the algorithm in the order of data availability

$$x_i^j = x_i^{j+1} - l_i^{i-j} * R_i^j$$
$$R_i^0 = x_i^i / l_i^i \qquad (3\text{-}4)$$
$$x_i^1 = b_i \qquad x_i = R_i^0 \qquad R_i^j = R_{i-1}^{j-1} \quad , \quad l_i^j = l_{i\ j}$$

for i=1 2 3 ,N , j=(N-1), 2 1 and i) j

Note that we have introduced a new variable R_i^j to propagate the data x_0 along the broadcast contour i-j = c This is because, the variable x_i^j has already been used in the i = c contour to denote the data involved in the iterative summation process

The localized DG obtained from the eq (3-4) is shown in Fig 3-2

Note that an algorithm is computable if and only if its DG contains no loops (an arc whose end points are the same node)

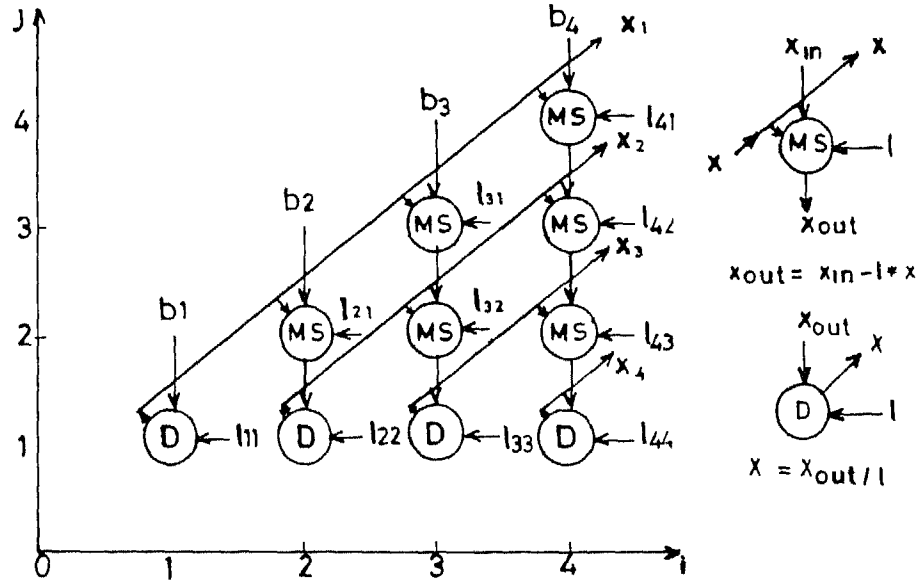


Fig 3 1 Globally Recursive DG for Lower Triangular Linear System Lx = B

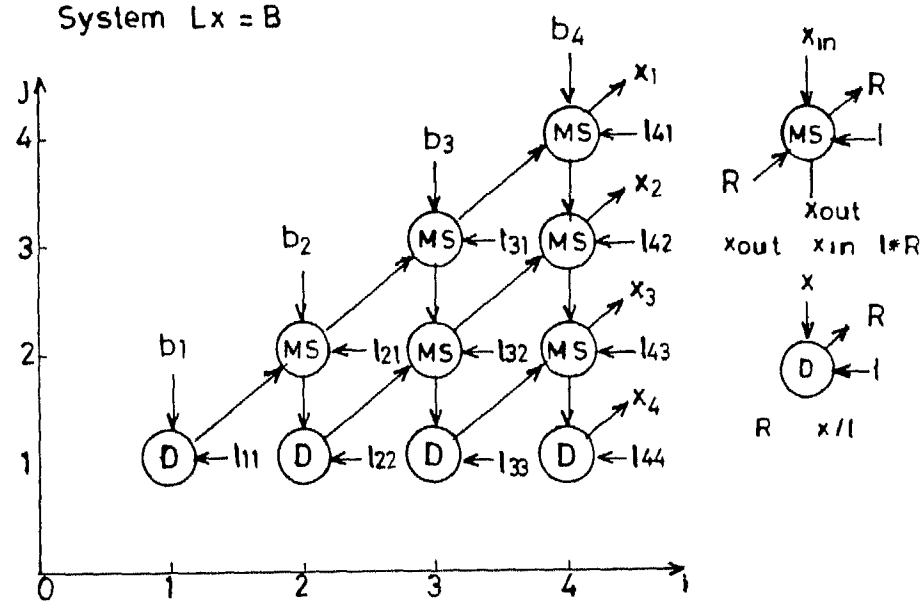


Fig 32 Locally Recursive DG for Lower Triangular Linear System Lx = B

or cycles (a chain of arcs directed in the same way whose end points are the same node)

The main objective of the studies reported in this chapter is to derive simple techniques for determining processor array implementation for a DG One simple and brute-force method for achieving this objective is to use a distinct processor for executing the computations at a particular index point of the DG This in general leads to a very inefficient use of the computational resources since each processor is active only for a constant period of time which could be a minute fraction of the time required for completing the algorithm To achieve better utilization of these resources it is necessary to reuse the same processor for handling a large number of computations In general the set of computations that are *allocated* to processors can be arbitrary disjoint subsets of the set of all computations However the problem of optimally scheduling' the computations that are *allocated* to a given processor becomes extremely difficult if the *allocation* is arbitrary To render this problem tractable one can restrict attention to linear (planar hyperplanar) partitions Here, a set of parallel lines (planes hyperplanes) are drawn through the index space of the DG so that all computations that correspond to index points that lie on the same line are handled by the same processor In this manner, the processor array itself can be obtained by projecting the DG along these lines (planes hyperplanes) onto lower dimensional lattice of

points known as the processor space However not all the projections of the DG will meet the regular communication local connectivity simple control and simple cell requirements of the systolic array architecture The determination of which projections will produce systolic arrays is the crux of this report

The DG for the algorithm under consideration is non-homogeneous consisting of MS cells and D cells A systolic solution produced from the projection of MS cells alone is called a *sub-systolic array* (SSA) as it is responsible for only a subset of computations Then the systolic array for LX = B is the combination of two sub-systolic arrays where one is produced from the projection of MS cells (SSA-1) and the other produced from the projection of D cells (SSA-2) The dimension of DG is n=2 Denote the homogeneous DG consisting of the MS cells as DG-1 and that of D cells as DG-2 Denote the index set of DG-i as I_i^n and its dependency vectors by d_{i1} d_{i2} Denote the matrix formed by dependence vectors of DG-i as D_i and that of dependence vectors from DG-i to DG-j as D_{ij}

Then, for the algorithm under consideration

$$I_1^2 = [\ (i\ j)\quad 2\le i\le 4\quad 2\le j\le 4\ ] = \begin{bmatrix} 2 & 3 & 4 & 3 & 4 & 4 \\ 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix}$$

$$I_2^2 = [\ (i\ j)\quad 1\le i\le 4\quad j = i\ ] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

DG-1 has two dependence vectors $d_{11} = (1\ 1)^t$ for pipelining the R variable and is obtained by joining the cells at

locations (2,2) and (3 3) and $d_{12} = (0\ -1)^t$ for pipelining the partial results and is obtained by joining the cells at location (3 3) and (3 2) Thus $D_1 = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$ DG - 2 has no dependence vectors Further $D_{12} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ and $D_{21} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Thus there are two considerations for mapping a DG onto an architecture

(1) In what ordering should the computations be scheduled

(2) To which cells the computations be allocated

The first step is the *scheduling* and the second step is the *allocation* The schedule function represents a mapping from the N-dimensional DG onto a one dimensional time space A schedule is based on a set of parallel hyperplanes and is represented by s pointing to the normal direction of the hyperplanes All the nodes in the hyperplane must be processed at the same timestep [18]

For a homogeneous DG, a schedule vector 's is said to be valid, if it satisfies the following condition

$$s^t * d_k > 0 \qquad (3\text{-}5a)$$

where d_k s are the dependence vectors of the DG

For a non-homogeneous DG-i a schedule vector s is said to be valid if it satisfies the following conditions

$$s^t * d_{ik} > 0 \quad \text{and} \quad s^t * d_{ijk} > 0 \qquad (3\text{-}5b)$$

where d_{ik} s are dependence vectors of DG-i and d_{ijk} s are dependence vectors from DG-i to DG-j Morever, this schedule

vector should be acceptable to all other DG-i s The above condition is to enforce *causality* in the permissible schedule vector Thus after finding a set of schedule vectors that satisfy the condition (3-5) a schedule vector which results in minimum computation time according to the following equation is chosen [12]

$$\text{computation time} \quad t = \frac{\text{Max } s^t * (i_1 - i_2) + 1}{\text{Min } s^t * d_{ik}} \qquad (3\text{-}6)$$

where i_1 and i_2 are any two index points of the DG and d_{ik} is any dependence vector of the DG The eq (3-6) represents the number of parallel hyperplanes s sweeping the index space I^n

For the algorithm under consideration $D_i = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$ and DG-2 has no dependence vectors Hence a schedule vector which is valid for DG-i is also valid for DG-2 It has been found that $s=(2\ -1)^t$, satisfies the condition (3-5) and results in minimum Computation time of 7 clock cycles

$$\text{Computationtime} = \frac{(2\ -1) * \left\{ \begin{bmatrix} 4 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\} + 1}{(2\ -1) * \begin{bmatrix} 0 \\ -1 \end{bmatrix}} = 7 \text{ clock cycles}$$

Thus the scheduling of the computations is done Now we need to *allocate* the computations to various processors

The projection of the MS cells onto the j axis is the same as the overlapping of all the column vectors onto one column The projection is said to be along the vector $(1\ 0)^t$ This vector

is indicated in the space by a line between two points with coordinates (0, 0) and (1, 0) The possible projection vectors for a 2-dimensional DG are p= $(1\ 0)^t$ $(-1\ 0)^t$ $(0\ 1)^t$ $(0\ -1)^t$ $(1\ 1)^t$ $(-1\ -1)^t$ $(1\ -1)^t$ and $(-1\ 1)^t$ Vectors with element values greater than 1 are not recommended as the resulting architecture needs many cells

In the discussion that follow some conditions have been formulated which are to be satisfied by a vector in order for it to become the valid projection vector

(1) The nodes on a equitemporal hyperplanes should not be projected onto the same cell This is because the cells on a equitemporal hyperplane have to be scheduled at the same timestep and if they are mapped onto a cell then that cell needs to do many computations at the same timestep which is *absurd* Hence for a projection vector to be valid

$$s^t * p > 0 \qquad (3\text{-}7)$$

For the algorithm under consideration, s = $(2\ -1)^t$ Hence all the possible projection vectors satisfies the above condition Note that if p is a valid projection vector, then -p will also be a valid projection vector even though it voilates the above condition Further if for some projection vector p $s^t*p=0$ then store this vector for future use with another schedule vector even though the new schedule vector may result in an increase in the execution time

(2) If the DG is non-homogeneous then the projection vector should not be equal to the data dependence vector in DG-i if there exists same kind of dependence vector from DG-i to DG-j as it demands a complex cell structure voilating the simple cell feature of systolic arrays

For the algorithm under discussion $D_i = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$ and $D_{i2} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ Hence $p = (0\ -1)^t$ is not valid Note that the final result of SSA-1 needs to be passed to SSA-2 Hence if we project the DG in $(0\ -1)^t$ direction the structure of the MS cells with partial results being fed back will not allow the final result to be passed to SSA-2

(3) In the case of non-homogeneous DG suppose that there exists a dependence vector d_1 from cell-w of homogeneous DG-i to cell-x of DG-j Further suppose that there exists same kind of dependence vector d_1 , within DG-i from cell-y to cell-z, where cell-y is in the neighbourhood of cell-w and in the possible projection direction Then vector joining cell-w and cell-y is not valid as the output line of cell-n which is obtained by projecting the cell-w onto cell-y needs to be connected to cell-x(of SSA-j) and cell-z (of SSA-i) depending upon the algorithm correctness This projection direction is invalid, because a DMUX is required at the output line of cell-n and hence complex control mechanism It is illustrated in Fig 3-3

For the algorithm under consideration there is a dependence vector $D_{12} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ from the cell with coordinates (2 2) and same kind of dependence vector $(0\ -1)^t$ exists within DG-1 from the cell with coordinates (3 3) Hence the projection vector $(1\ 1)^t$ which maps the cells at coordinates (2 2) and (3 3) onto one cell is not valid Similarly the projection vector $(1\ -1)^t$ which maps the cells at coordinates (3 3) and (4 2) is also *not valid*

So far the projection vector $(1\ 0)^t$ is only *not invalid* There are some more conditions of this type which are to be satisfied by a projection vector to be valid These conditions are described here although the projection vector $(1\ 0)^t$ does not voilate any of these conditions

(4) If the DG is non-homogeneous and a dependence vector d_1 is present from DG-j to DG-i and also in DG-i,then that dependence vector 'd_1 is 'not a valid projection vector This is because a variable *'computed'* in SSA-j is being allowed to *'stay'* in SSA-i which needs a complex control mechanism as opposed to the *'simple control* requirement of systolic arrays

For the algorithm under consideration $D_{21} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and same kind of dependence vector $(1\ 1)^t$ is present within DG-1 Hence $(1\ 1)^t$ is not a valid projection vector

(5) In the case of non-homogeneous DG suppose that there exists a dependence vector d_1 from cell-a of DG-i to cell-b of DG-j Further suppose that there exists same kind of dependence vector from cell-c to cell-d within DG-j where cell-d is in the neighbourhood of cell-b and is in the possible projection direction Then the vector joining cell-b and cell-d is not a valid projection vector as the input line of cell-n which is obtained by projecting the cell-b onto cell-d needs to be connected to the cell-a (of SSA-i) and cell-c (of SSA-j) It is illustrated in Fig 3-4

For the algorithm under consideration there is a dependence vector $D_{2i} = \begin{bmatrix} i \\ i \end{bmatrix}$ from the cell with coordinates (2 i) and same kind of dependence vector $(i\ i)^t$ is also present within DG-i from the cell with coordinates (2 2) Hence the projection vector $(0\ -i)^t$ which maps the cells at coordinates (3 2) and (3,3) onto one cell is not valid

Thus for the lower triangular linear systems of equations LX=B the projection vector $(i\ 0)^t$ is the only valid projection vector Constructing the systolic array, with this information is the topic of the next section

3 3 ARCHITECTURAL SPECIFICATION

The systolic array for LX = B algorithm can be obtained by projecting the corresponding DG in $(i\ 0)^t$ direction This

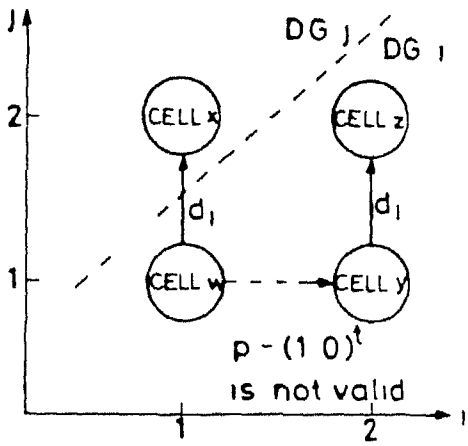


Fig 3-3 Violation of condition 3

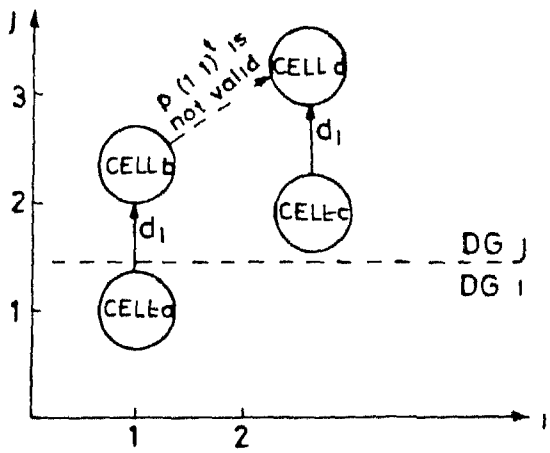


Fig 3 4 Violation of condition 5

graph-based projection procedure may be formally described in terms of algebraic transformations For this first we need to find the processor basis A corresponding to each valid projection vector to identify the processors This processor basis A is denoted by nX(n-1) matrix and is orthogonal to the projection vector p Obviously there exists many vectors corresponding to the processor basis that are perpendicular to the projection vector p But one can choose any set of (n-1) vectors perpendicular to p in order to identify the processors because the choice of the basis does not change the topology of the array but only modifies the labels assigned to the processors

For each remaining valid' projection vector find a set of vectors A_m perpendicular to the projection vector p with element values being restricted to -1, 0 and 1 Then again form a new set of vectors that satisfy the 'partitioning [1] condition given below

$$A_m * d_{1k} \geq 0 \qquad (3\text{-}8)$$

where d_{1k}'s are dependence vectors of DG-1 The above condition is to be satisfied by all homogeneous DGs and is to ensure that the communication between the links is not cyclic [12] If there are atleast (n-1) vectors which satisfies the above

---

[1] For details about the partitioning schemes that is mapping big sized problem onto the resulting small architecture', refer the Appendix

condition then the resulting architecture is partitionable and a set of (n-1) vectors is chosen as the processor basis A Otherwise the resulting architecture is non-partitionable

For the algorithm under consideration $(1\ 0)^t$ is the only valid projection vector There are two vectors namely $(0\ 1)^t$ and $(0\ -1)^t$ with element values being restricted to -1 0 and 1 and perpendicular to this projection vector

But both of these vectors does not satisfy the eq (3-8) Hence the resulting architecture is not partitionable The dimension of the DG under consideration is n=2 Hence processor basis A consists of only one A_m vector The vector $(0\ 1)^t$ is chosen as the processor basis A

Now form a geometric transformation matrix $T=\begin{bmatrix} S^t \\ A^t \end{bmatrix}$ The mapping T should be bijection i e the mapping T should be 'onto and 'one-to-one The necessary and sufficient condition for T to be bijection is that it should be non-singular i e detT≠0 If detT=0 then choose another set of (n-1) vectors as basis and proceed further For the algorithm under consideration $T=\begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix}$ and detT=2 With this information the architecture is obtained as illustrated further

FORMING THE ARCHITECTURE

Obtain the new index set and transformed dependencies by

applying the geometric transformation matrix T to the index set and dependence vectors of the DG respectively Then with this information the systolic architecture is constructed as follows

The last (n-1) rows of the transformed index set specifies the labels of the processors Hence the cells are sketched by noting the different labels of the processors Then links between the cells are obtained from transformed dependencies Note that the first row of the transformed dependencies specifies the amount of external delay on that link and the remaining rows are difference vectors of the labels of the processors and represent the physical link between the processors For example a link $\begin{bmatrix} a \\ b \end{bmatrix}$ means there is an external delay of (a-1) clock cycles on link b which exists from cell with label l to a cell with label m where b= m-l Data timings are extracted from the transformed index set in which the first row specifies the timestep at which the computations are scheduled and the remaining rows specifies the processors to which the computations are allocated

For the algorithm under consideration there are two index sets I_1^n and I_2^n and the dimension n of the DG is 2 The transformed index sets and dependence vectors are printed below

$$T*I_1^2 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 4 & 3 & 4 & 4 \\ 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 & 3 & 5 & 4 \\ 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix}$$

$$T*I_2^2 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$T*D_1 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$T*D_{12} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \text{and} \qquad T*D_{21} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Thus there are three MS cells according to the last row of the transformed index set of I_1^2 and one D cell according to the last row of the transformed index set of I_2^2 From the transformed dependencies of D_1 it can be found that in sub-systolic array-1 (SSA-1), there exists a link (1) from the cell with label 2 to the cell with label 3 with no external delay and same kind of link exists from the cell with label 3 to the cell with label 4 Further in SSA-1 there exists a link (-1) from the cell with label 3 to the cell with label 2 with no external delay and same kind of link exists from the cell with label 4 to the cell with label 3 Transformed dependencies of D_{12} indicates that there exists a link (-1) from the cell with label 2 of SSA-1 to the cell with label 1 of SSA-2 There exists a link (1) from the cell with label 1 of SSA-2 to the cell with label 2 of SSA-1 according to the transformed dependencies of D_{21} To obtain data timings consider, for example the computation indexed by (2 1) The tranformed coordinates are (3 1) meaning that the computation time is 3 and the label of the cell is 1 Notice that the tranformed coordinates are offset by some initial values

The resulting architecture for *solution of lower triangular linear systems* $Lx = b$ is shown in Fig 3-5

The throughput which is defined as the number of output components per clock cycle is equal to half i e one output sample per two clock cycles for the architecture in Fig 3-5 Initial offset which is defined as the time that has to be elapsed till the computations start is (N-1) = 3 clock cycles for this architecture Its computation time is (2*N-1) = 7 clock cycles Final offset which is defined as the extra time needed to clear all the output components in the pipeline when the last computation is over is (N-1) = 3 clock cycles for this architecture Thus the total execution time required which is obtained by adding the initial and final offsets to the computation time is 13 clock cycles Further note that this architecture is non-partitionable thereby requiring N cells to construct

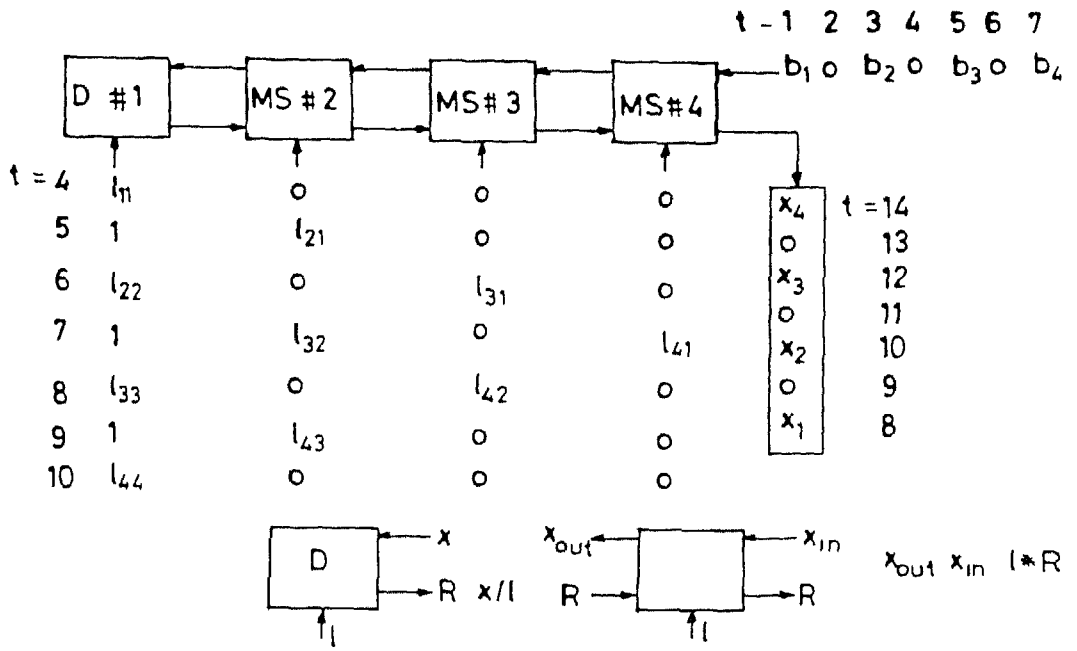


Fig 3 5 Systolic solution for the lower triangular linear systems $LX=B$ p (1 0)^t

CHAPTER4
DESIGN TOOLS AND RESULTS

I ever loved to see everything in circles and squares

-CERVANTES

A software package ADSA is developed for Automatic Design of Systolic Arrays Section 4 1 describes the design facility The systolic solutions for upper triangular linear systems of equations matrix-matrix multiplication discrete fourier transform LU decomposition FIR filter recursive convolution and IIR filter are given in section 4 2

4 1 DESIGN FACILITY

The block diagram of the design facility is shown in figure A The input to the ADSA program is the dimension n indices and dependence information of the DG For a homogeneous DG the indices and dependence information of the DG consists of the index set and dependence vectors of the DG In case of non-homogeneous DG this information comprises of the index sets of various DGs and the dependence vectors within each DG-i and from each DG-i to DG-j

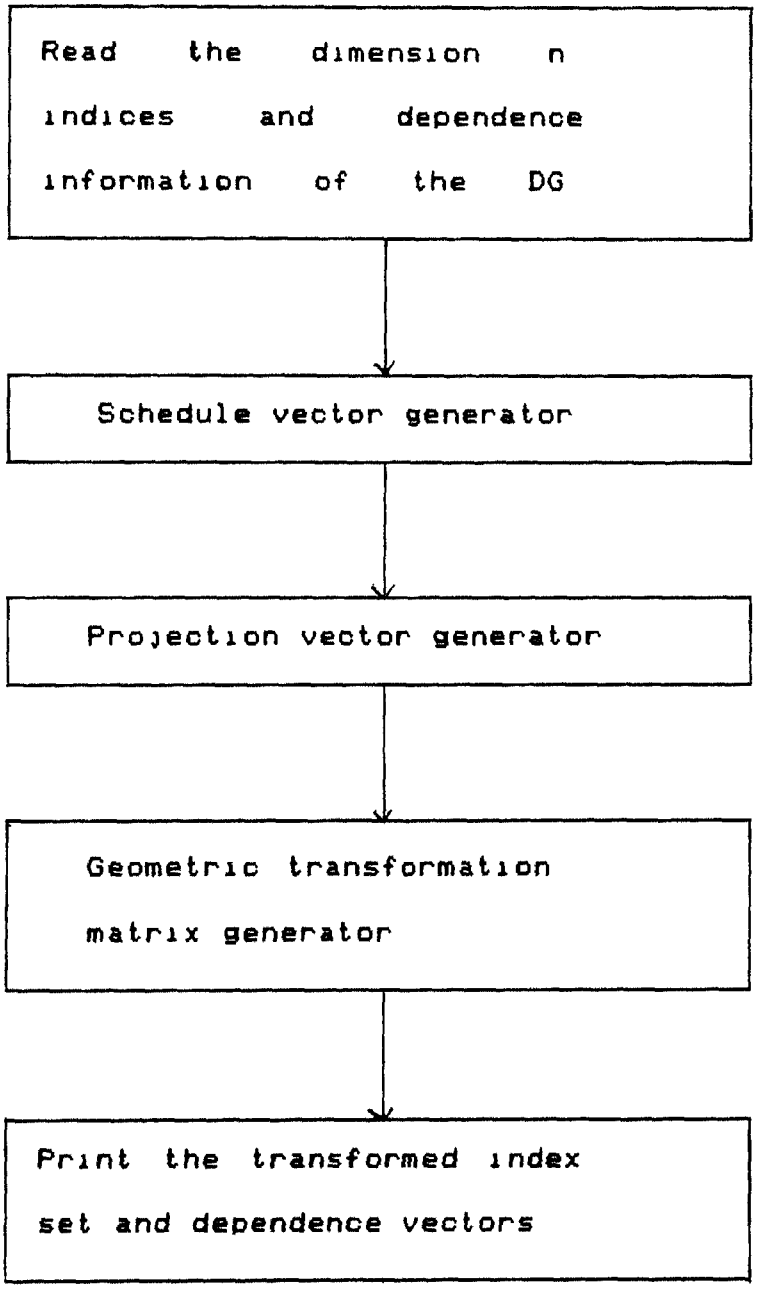


Figure A The ADSA system block diagram

Schedule vector generator This part of the program finds a set of schedule vectors that satisfy the eq (3-5) and then chooses a schedule vector which results in minimum execution time according to the eq (3-6) If no schedule vector exists then it asks the user to modify either the dependence information of the DG or to use a different algorithm of the same problem

Projection vector generator This part of the program first finds all the projection veectors that satisfy the condition $s^t * p > 0$ If for some projection vector p $s^t * p = 0$ then this vector is stored for to be used with another schedule vector

If the DG is non-homogeneous, then it removes all the projection vectors that voilate the conditions given below

(1) Projection vector should not be equal to the dependence vector from DG-i to DG-j if there exists same kind of dependence vector within DG-i

(2) Projection vector should not be equal to the dependence vector from DG-i to DG-j if there exists same kind of dependence vector within DG-j

(3) If there is a dependence vector from cell-a (an index of the DG) of DG-i to cell-b of DG-j and same kind of dependence vector exists within DG-i from cell-c to cell-d then the projection vector which maps the cell-a onto cell-c is not valid

(4) If there is a dependence vector from cell-a of DG-i to cell-b of DG-j and same kind of dependence vector exists within

DG-j from cell-c to cell-d then the projection vector which maps the cell-b onto cell-d is not valid
If no projection vector exists then it asks the user to either modify the algorithm or use a different algorithm of same problem

Transformation matrix generator This part of the program finds the processor basis A which is a set of (n-1) vectors perpendicular to the projection vector and hence geometric transformation matrix T for each valid projection vector

Finally the program prints the transformed index set and dependence vectors from which the architecture is constructed

This software is developed only for the purpose of research and is needed to be generalized as the maximum dimension n of the DG being allowed is 3 (assuming that a higher dimensional DG can be split into 3D DGs and use multiprojection [1] to obtain a planar systolic array) and the number of non-homogeneous DGs being allowed is 3

The input to the ADSA program is the data dependence vectors and index set of the DG and this information is obtained manually This step may be automated but more research is needed for this In general the solution to a problem may

---

[1] See the Appendix for discussion on 'multiprojection'

be provided by a number of different algorithms Each algorithm may lead to a different DG and hence different architectures For example the discrete fourier transform problem can be formulated as matrix-vector multiplication algorithm Further an entirely different DG can be obtained if we apply Horner s nested rule to the same problem

At present the architecture is constructed manually using the transformed index set and dependencies This step can also be automated Further the step of finding an optimal systolic solution can also be automated by computing an optimization index such as time area etc The total required execution time is available and the area results from the number of processors and the complexity of each processor The concepts of multiprojection partitioning etc can be readily implemented

The system is used to design systolic arrays for several known problems The system has reproduced known systolic solutions for these algorithms and in some cases has produced several new solutions

4 2 RESULTS

This method for constructing systolic arrays is applied to several problems and results are reported Each algorithm is translated into locally recursive equations from which the DG is obtained For each algorithm the method is applied to small

problem size The selected problem size is the smallest size that represents the algorithm That is the DG generated from two different problem sizes should differ only in size and not in the number of subsets of computations For each algorithm the valid schedule and projection vectors are reported Geometric transformation matrices data flow timing and data movement are also reported

4 2 1 UPPER TRIANGULAR LINEAR SYSTEMS OF EQUATIONS UX=B

The algorithm for the solution of the upper triangular linear systems of equations UX=B where U is an NXN upper triangular matrix and X and B are NX1 vectors is described as

$$x_i = (b_i - \sum_{j>i} u_{ij} * x_j) / u_{ii} \quad \text{for } i=N \quad 2\ 1 \qquad (4\text{-}1)$$

For this algorithm the calculations for N=4 are explicitly expressed as follows

$$\begin{aligned} x_4 &= (b_4) / u_{44} \\ x_3 &= (b_3 - u_{34} * x_4) / u_{33} \\ x_2 &= (b_2 - u_{24} * x_4 - u_{23} * x_3) / u_{22} \\ x_1 &= (b_1 - u_{14} * x_4 - u_{13} * x_3 - u_{12} * x_2) / u_{11} \end{aligned} \qquad (4\text{-}2)$$

In this algorithm the elements of vector X are calculated by back-substitution That is x_4 is calculated first and then used in the calculation of x_3 then both of them are used in the calculation of x_2 and so on The natural order of performing these calculations is in the order of data availability of the elements of vector X The following forward recurrence equations assume this natural order

$$\begin{aligned} x_i^j &= x_i^{j+1} - u_i^j * R_i^j \\ R_i^0 &= x_i^1 / u_i^0 \\ x_i^1 &= b_{N+1-i} \qquad x_i = R_N^{i-1} \\ R_i^j &= R_{i-1}^{j-1} \qquad \text{and} \qquad u_i^j = u_{N+1-i\ N+1-(i-j)} \end{aligned} \qquad (4\text{-}3)$$

for $i=1,2 \quad N \qquad j=(N-i) \quad 2\ 1 \quad \text{and} \quad i)j$

Note that a new variable R_1^J has been introduced to propagate the data x_C along the broadcast contour $i-j = m$ This is because the variable x_1^J has already been used in the $i = m$ contour to denote the data involved in the iterative summation process The localized DG obtained from eq (4-3) is shown in the figure 4-1

Since the DG consists of two types of cells the systolic solution will be a non-homogeneous array Let the DG consisting of MS cells be denoted by DG-1 and the DG consisting of D cells be denoted by DG-2 It has been found that there exists only one valid projection vector namely $(1\ 0)^t$ and the schedule vector $(2\ -1)^t$ results in minimum computation time of seven clock cycles Thus the processor basis A corresponding to this projection vector is $(0\ 1)^t$ Hence the geometric transformation matrix $T=\begin{bmatrix} s^t \\ A^t \end{bmatrix}=\begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix}$ The transformed index sets and dependencies are printed below

$$T * I_1^2 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 4 & 3 & 4 & 4 \\ 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 & 3 & 5 & 4 \\ 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix}$$

$$T * I_2^2 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$T * D_1 = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$T * D_{12} = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$T * D_{21} = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The architecture is shown in figure 4-2 which is constructed from the transformed index sets and dependencies Note that the last row of the transformed index set specify the labels of the processor Thus there exists three MS cells and one D cell The dependence vector $\begin{bmatrix}0\\-1\end{bmatrix}$ of variable x is transformed into a link (-1) with no external delay Thus there exists a link (-1) within sub-systolic array-1 (SSA-1) consisting of MS cells from the cell with label 3 to the cell with label 2 and from the cell with label 4 to the cell with label 3 Similarly various links are obtained by noting that the transformed dependencies $T*D_i$ indicates the links that exists within SSA-i where as $T*D_{ij}$ indicates the link that exists from SSA-i to SSA-j After sketching the architecture the data timings are extracted from the transformed index set in which the first row specifies the time at which a computation is scheduled and the remaining rows specify the processor to which that computation is allocated

This architecture has an initial offset of (N-1)=3 clock cycles Its computation time is (2*N-1)=7 clock cycles Since there is only one output component per two clock cycles the throughput is half Note that this architecture is non-partitionable, hence the number of cells required depends upon the size N of the problem

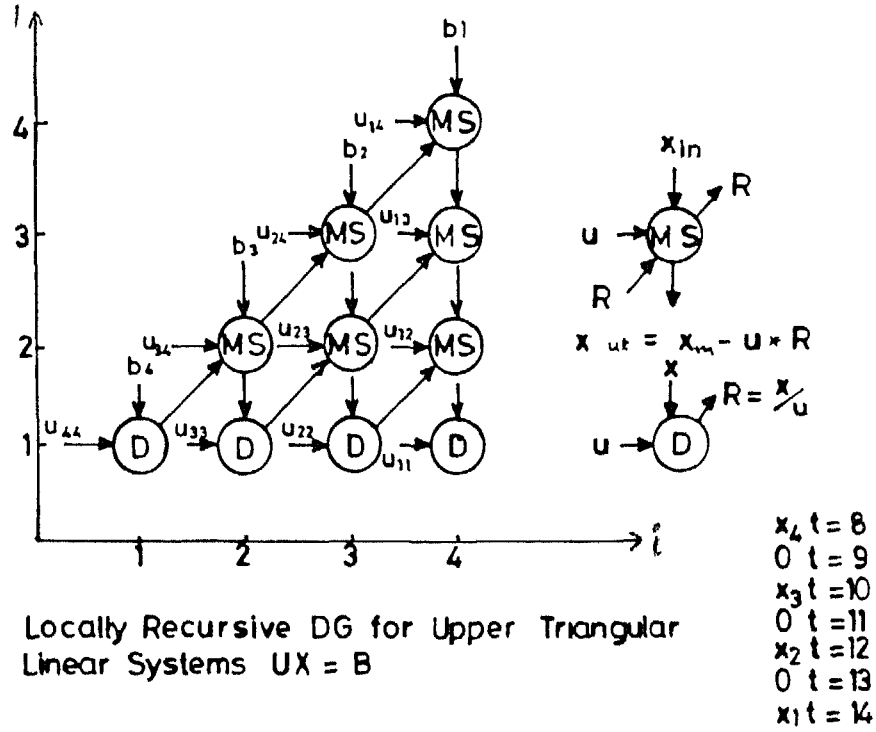


Fig 4.1 Locally Recursive DG for Upper Triangular Linear Systems UX = B

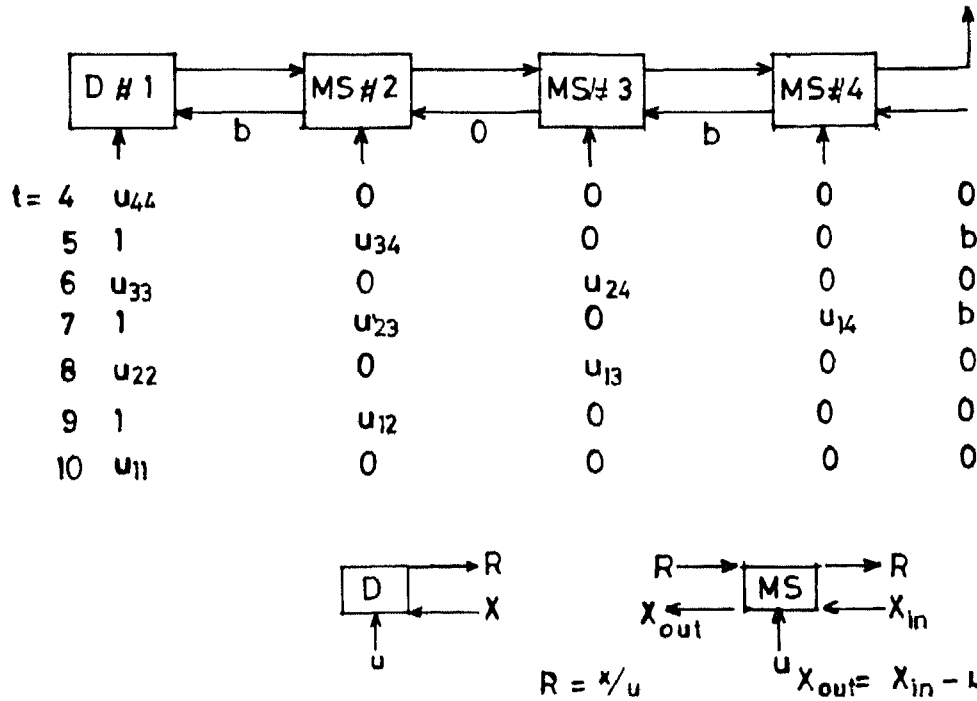


Fig 4.2 Systolic Solution for Upper Triangular Linear Systems UX=B

4 2 2 MATRIX-MATRIX MULTIPLICATION

Consider matrix-matrix multiplication algorithm C= A*B where $A(a_{ij})$ is NXM matrix $B(b_{ij})$ is MXQ matrix and $C(c_{ij})$ is NXQ matrix This algorithm is described by

$$c_{ij}=\sum_{k=1}^{M} a_{ik} * b_{kj} \quad \text{for } i=1\ 2\ \ N \text{ and } j=1\ 2\ \ Q \qquad (4\text{-}4)$$

The locally recursive version of this algorithm is

$$\begin{aligned} c_{ij}^{k} &= c_{ij}^{k-1} + a_{ij}^{k} * b_{ij}^{k} & c_{ij} &= c_{ij}^{M} \\ c_{ij}^{0} &= 0 & a_{ik} &= a_{ii}^{k} \\ a_{ij}^{k} &= a_{i(j-1)}^{k} & b_{kj} &= b_{ij}^{k} \\ b_{ij}^{k} &= b_{(i-1)j}^{k} & & \end{aligned} \qquad (4\text{-}5)$$

The DG obtained from eq (4-5) is shown in figure 4-3 Since the DG contains only one type of cell the systolic solutions are homogeneous arrays Further each cell does the *multiply and accumulate* (MAC) operation and hence is not indicated For N=3 M=3 and Q=3, the index set and dependence vectors of the DG are given below

$$I^{3} = \begin{bmatrix} 1&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2&3&3&3&3&3&3&3&3&3 \\ 1&1&1&2&2&2&3&3&3&1&1&1&2&2&2&3&3&3&1&1&1&2&2&2&3&3&3 \\ 1&2&3&1&2&3&1&2&3&1&2&3&1&2&3&1&2&3&1&2&3&1&2&3&1&2&3 \end{bmatrix}$$

$$D = \begin{bmatrix} 0&1&0 \\ 1&0&0 \\ 0&0&1 \end{bmatrix}$$

The first dependence vector $(0\ 1\ 0)^{t}$ is due to the pipelining of the variable a, the second vector is due to the pipelining of the variable b and the last vector corresponds to the accumulation of partial results that is the variable c

The procedures were applied to this DG and found that there exist thirteen valid projection vectors The processor basis A corresponding to each valid projection vector is also obtained Geometric transformation matrices corresponding to the valid projection vectors are listed in table 4-1 Note the various schedule vectors used Architectural details are also listed in this table Final offset is defined as the extra time needed to clear all output components in the pipeline when the last computation is over The total execution time required is found by adding initial and final offsets to the computation time Further note that all systolic solutions are partitionable except the solutions 4 5 6 and 7 The systolic solutions corresponding to the thirteen valid projection vectors are shown in figure 4-4 through 4-16 The architecture of figure 4-4 corresponding to the projection vector $(1\ 0\ 0)^t$ is obtained as follows

The geometric transformation matrix corresponding to the projection vector $(1\ 0\ 0)^t$ is $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ The schedule vector $(1\ 1\ 1)^t$ results in minimum execution time of (N+M+Q-2)=7 clock cycles The transformed index set and dependencies are given below

$$T*I^3=\begin{bmatrix} 3 & 4 & 5 & 4 & 5 & 6 & 5 & 6 & 7 & 4 & 5 & 6 & 5 & 6 & 7 & 6 & 7 & 8 & 5 & 6 & 7 & 6 & 7 & 8 & 7 & 8 & 9 \\ 2 & 3 & 4 & 3 & 4 & 5 & 4 & 5 & 6 & 2 & 3 & 4 & 3 & 4 & 5 & 4 & 5 & 6 & 2 & 3 & 4 & 3 & 4 & 5 & 4 & 5 & 6 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}$$

$$T*D = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The last two rows of the transformed index set specify the *labels* of the processors Thus the architecture needs nine processors The dependence vector $(0\ 1\ 0)^t$ of variable a is transformed into a link $\begin{bmatrix}1\\1\end{bmatrix}$ with no external delay Thus there is a link from the cell with label $\begin{bmatrix}2\\1\end{bmatrix}$ to cell with label $\begin{bmatrix}3\\2\end{bmatrix}$ from the cell with label $\begin{bmatrix}3\\2\end{bmatrix}$ to cell with label $\begin{bmatrix}4\\3\end{bmatrix}$ and so on to pipeline the variable a Further the dependence vector $(1\ 0\ 0)^t$ of variable b is transformed into a self loop with no external delay Hence the elements of variable b will *stay* in the cells The dependence vector $(0\ 0\ 1)^t$ of variable c is transformed into a link $\begin{bmatrix}1\\0\end{bmatrix}$ with no external delay The timestep at which a computation is scheduled and to which processor it is allocated are noted from the transformed index set Thus the data timings are extracted which completes the architecture

The transformed index set and dependencies of various valid projection vectors are not reported here due to prolixity The veracity of the architectures in figure 4-5 through 4-16 can be verified by forming the transformed index set and dependencies and following the procedure illustrated for the projection vector $(1\ 0\ 0)^t$

The best systolic solution for matrix-matrix multiplication depends on the specific application From the results indicated in table 4-1, for minimum chip area, the best solutions are 1-3 For minimum execution time the best solutions are 8-10

Optimal solutions based on AT or AT^2 optimization can also be selected Note that the total required execution time for solutions 1-3 depends on the implementation of the loading / unloading process and chip area for solutions 11-13 inculdes the implementation of delay elements between the cells

Table 4-1 Comparison of architectures for Matrix-Matrix multiplication

	Projection vector	Transformation matrix	Number of cells	Initial offset	Computation time	Final offset
1	$(1\ 0\ 0)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	9	-	7	-
2	$(0\ 1\ 0)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	9	-	7	-
3	$(0\ 0\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	9	-	7	-
4	$(1\ 1\ 0)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$	15	2	7	2
5	$(1\ 0\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	15	2	7	2
6	$(0\ 1\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$	15	2	7	2
7	$(1\ 1\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{bmatrix}$	19	2	7	2
8	$(1\ 1\ -1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$	19	-	7	-
9	$(1\ -1\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$	19	-	7	-
10	$(-1\ 1\ 1)^t$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$	19	-	7	-
11	$(1\ -1\ 0)^t$	$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	15	2	9	-
12	$(1\ 0\ -1)^t$	$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$	15	2	9	-
13	$(0\ 1\ -1)^t$	$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	15	2	9	-

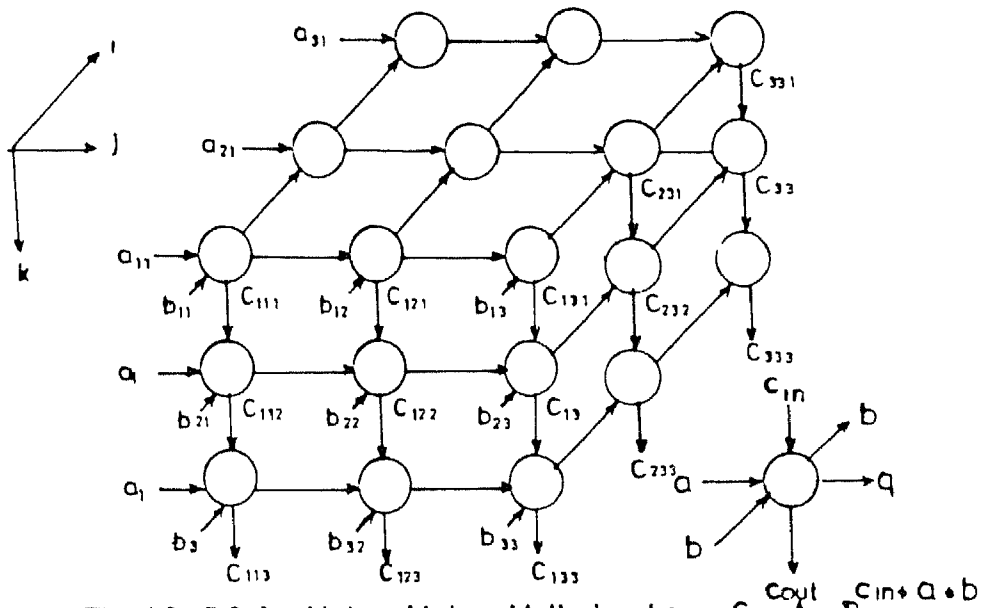


Fig 4.3 DG for Matrix-Matrix Multiplication C = A * B

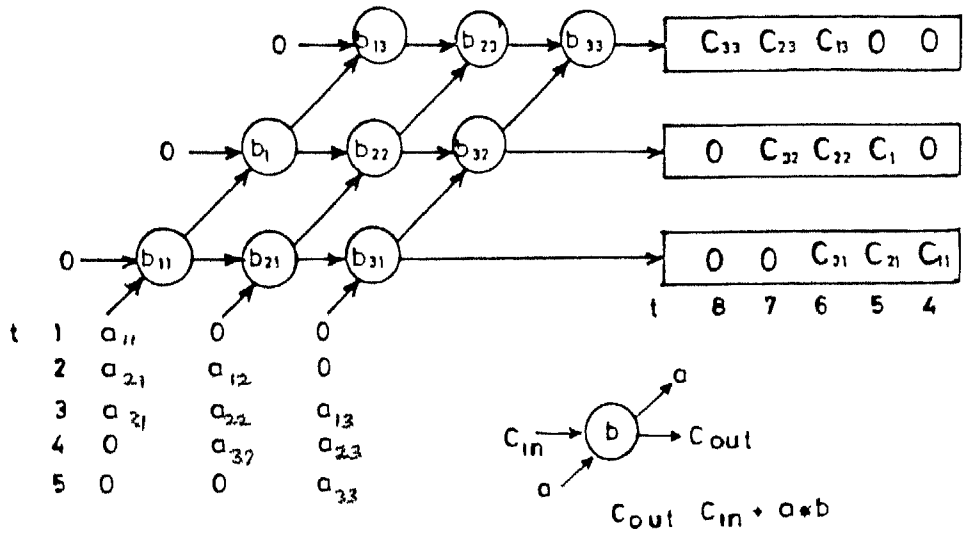


Fig 4.4 Systolic Solution for Matrix Matrix Multiplication C= A* B p=(1 0 0)

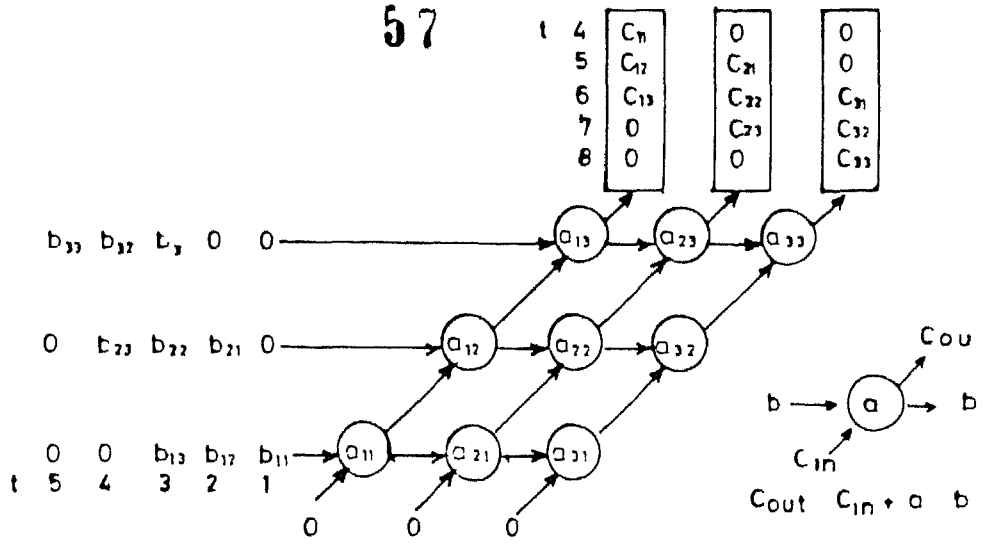


Fig 4 5 Systolic Solution for Matrix-Matrix Multiplication C = A * B p=(010)^t

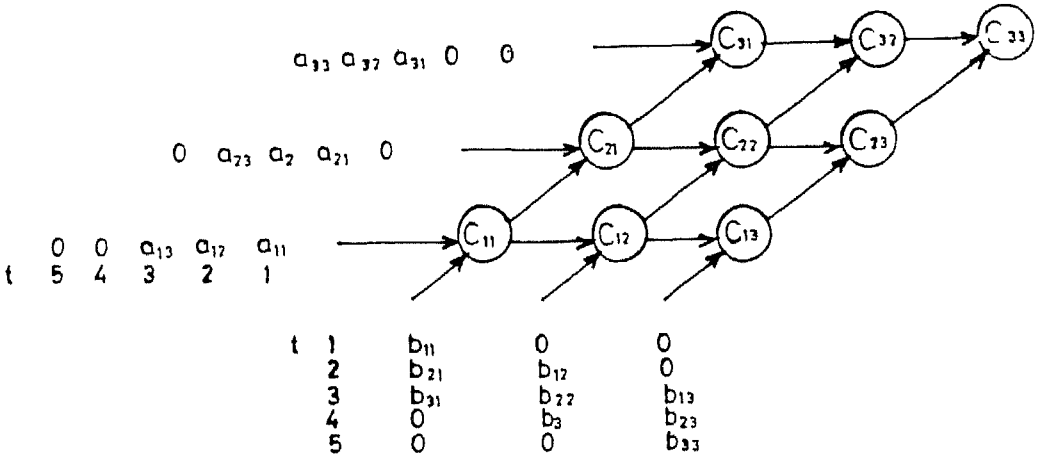


Fig 4 6 Systolic Solution for Matrix Matrix Multiplication
C = A * B p = (001)^t

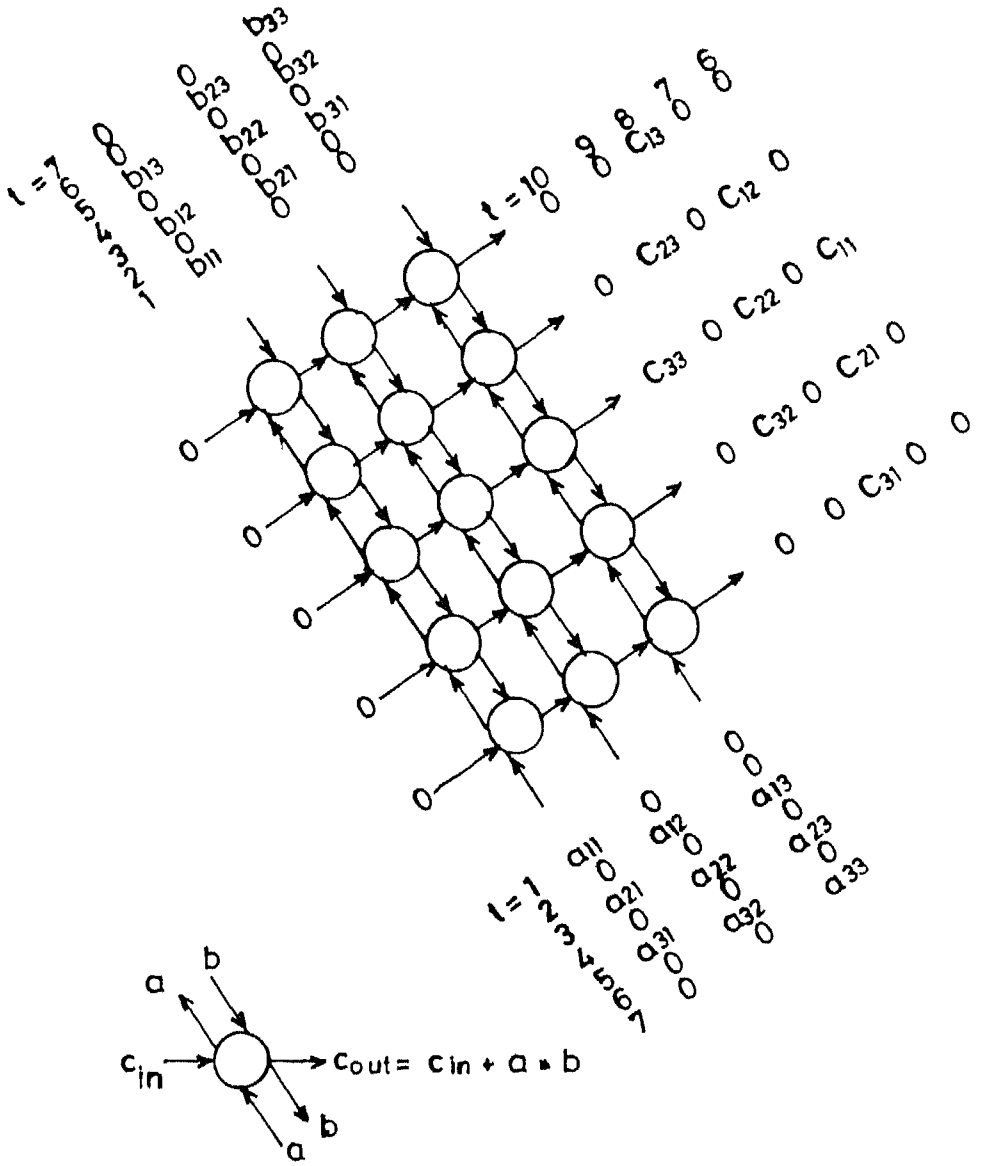


Fig 4 7 Systolic Solution for Matrix Matrix Multiplication $C = A * B$, $p = (110)^t$

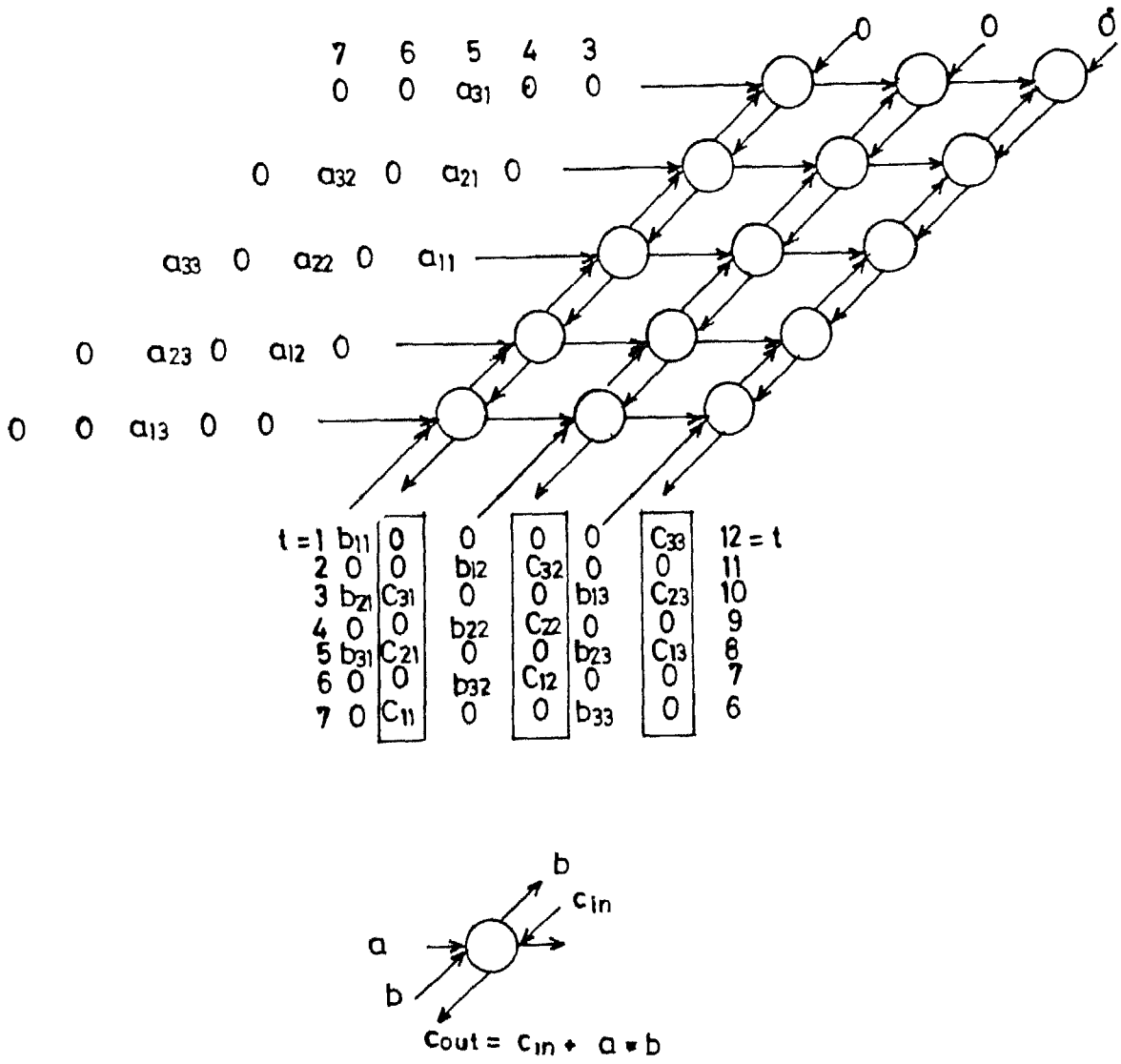


Fig 4 8 Systolic Solution for Matrix -Matrix Multiplication C=A*B p=(101)^t

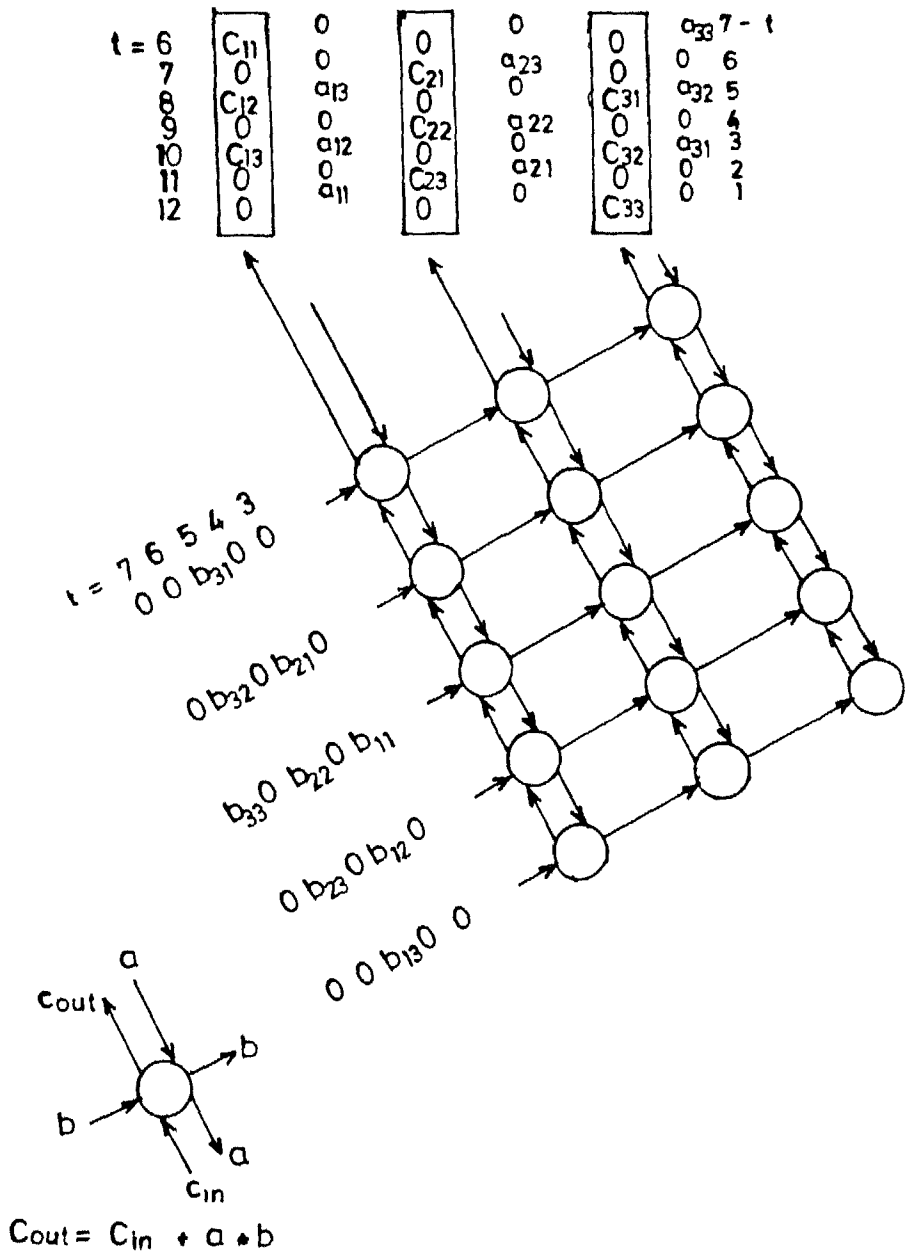


Fig 4 9 Systolic Solution for Matrix-Matrix Multiplication C=A*B p=(0 1 1)^t

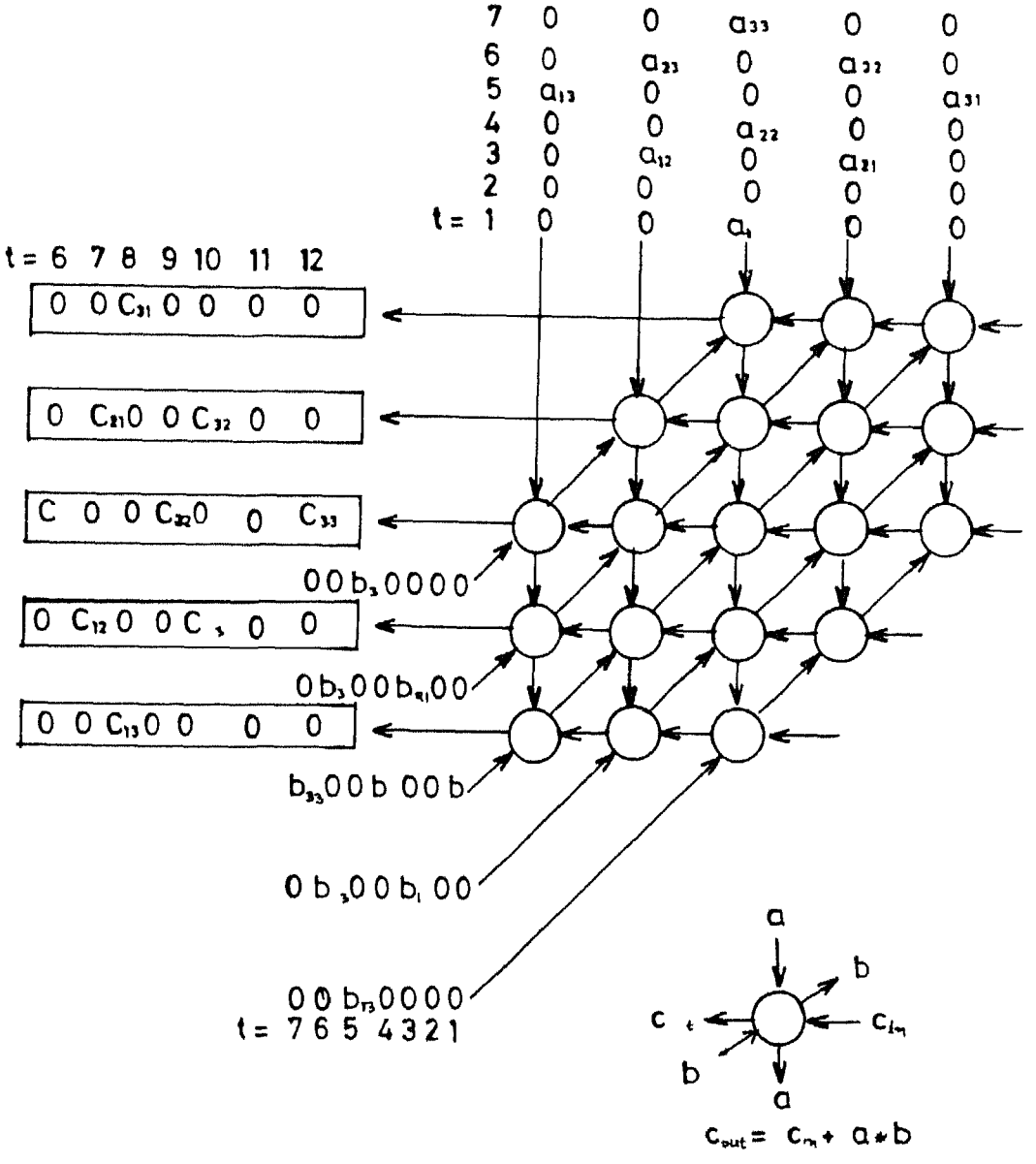


Fig 4 10 Systolic Solution for Matrix - Matrix Multyplication $C=A*B$ $p=(111)^t$

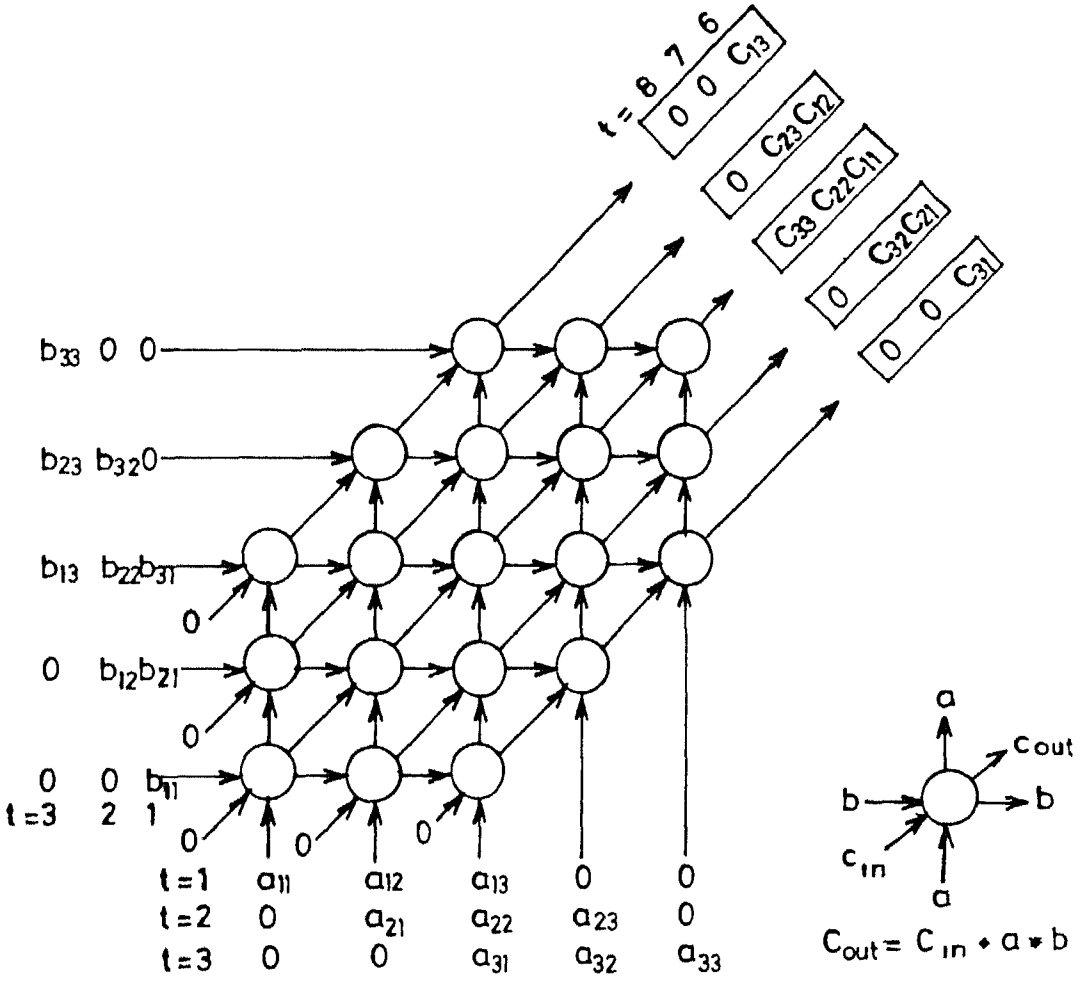


Fig 4.11 Systolic Solution for Matrix - Matrix Multiplication C = A * B $p=(11-1)^t$

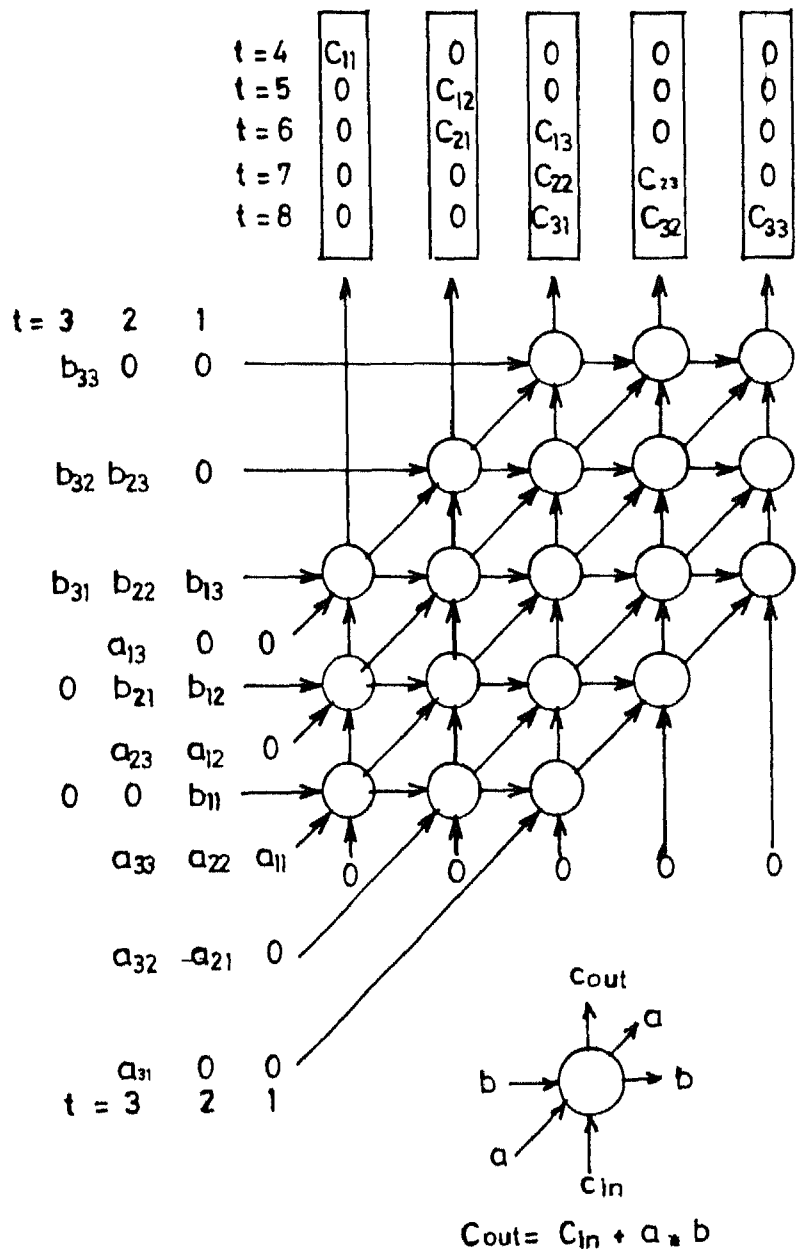


Fig 4.12 Systolic Solution for Matrix Matrix Multiplication $C=A*B$ $p=(1\ 1\ 1)^t$

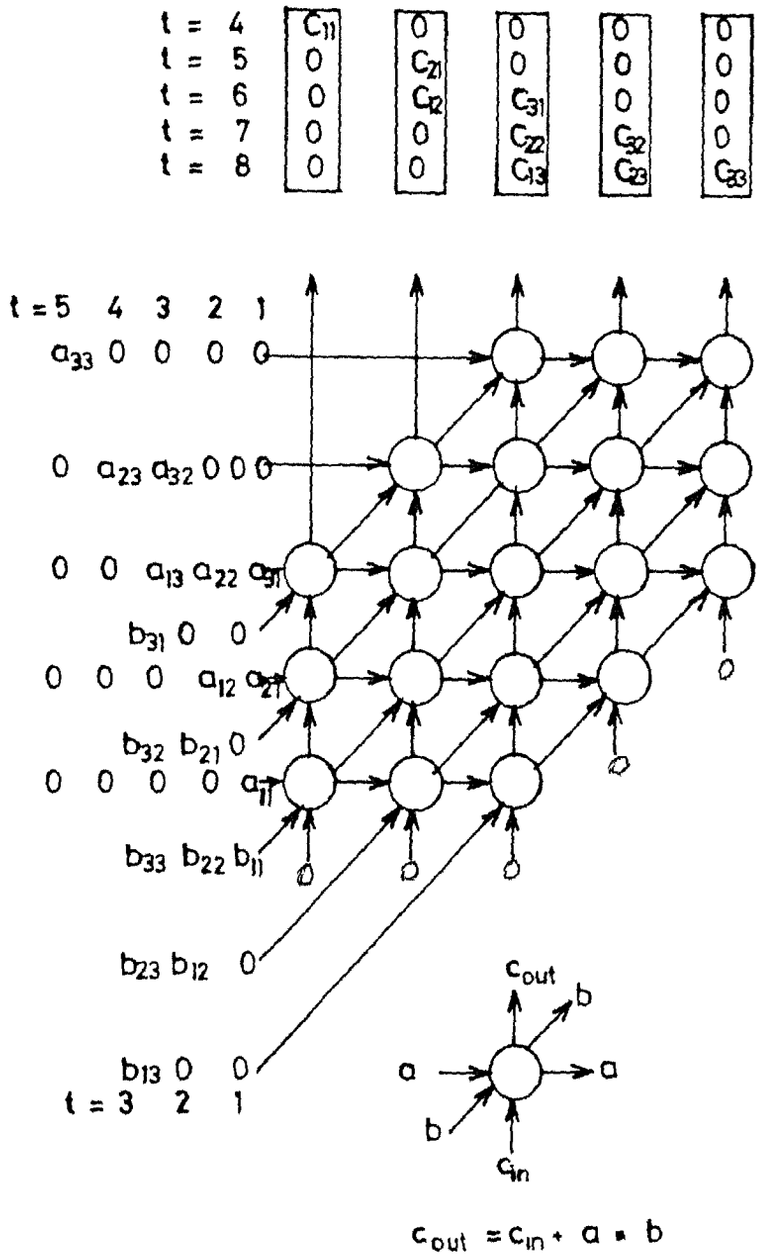


Fig 4.13 Systolic Solution for Matrix-Matrix Multiplication C=A B $p=(-111)^t$

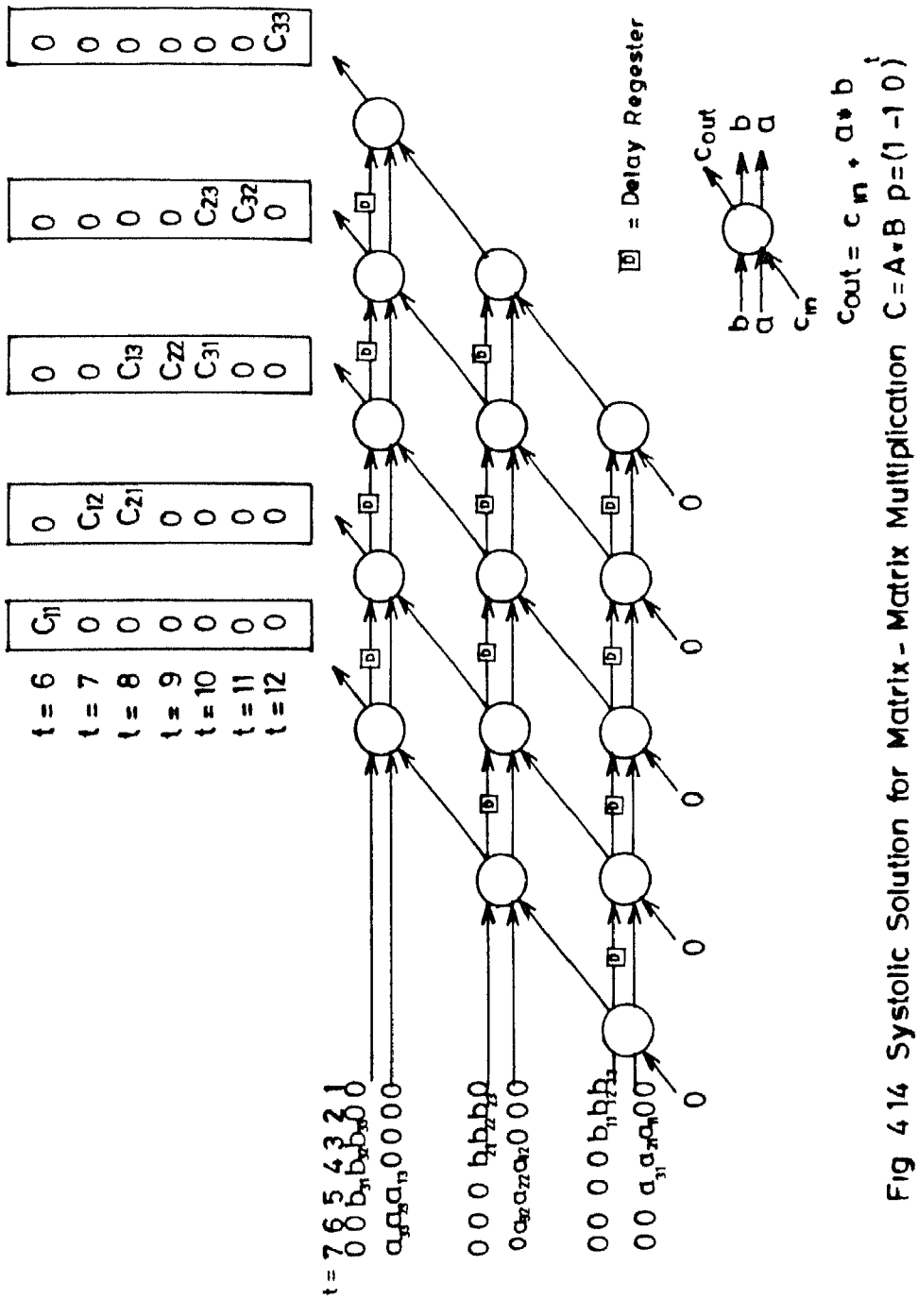


Fig 4.14 Systolic Solution for Matrix - Matrix Multiplication $C = A * B$ $p = (1\ -1\ 0)^t$

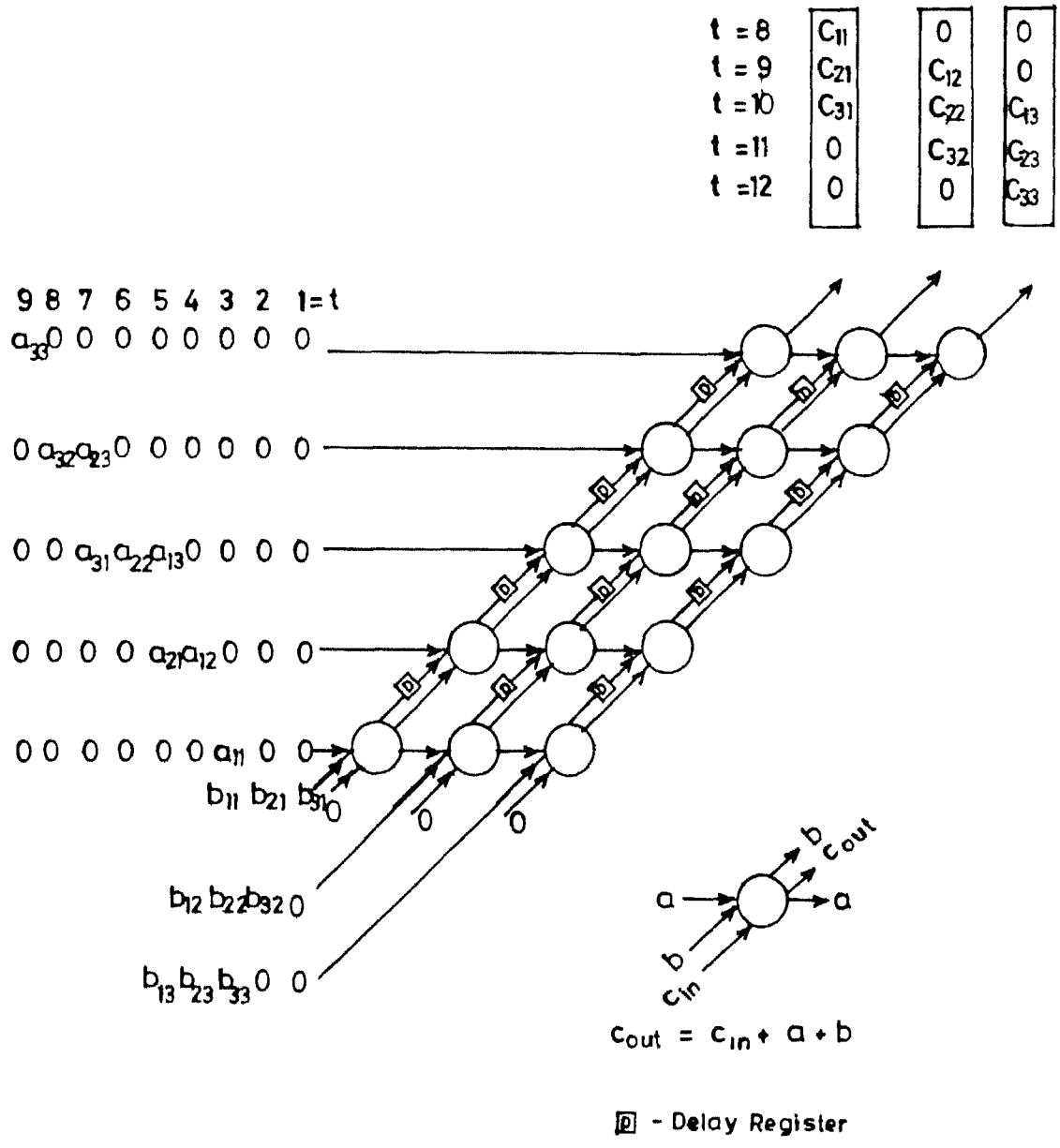


Fig 4.15 Systolic Solution for Matrix -Matrix Multiplication $C = A * B$ $p = (10\text{-}1)^t$

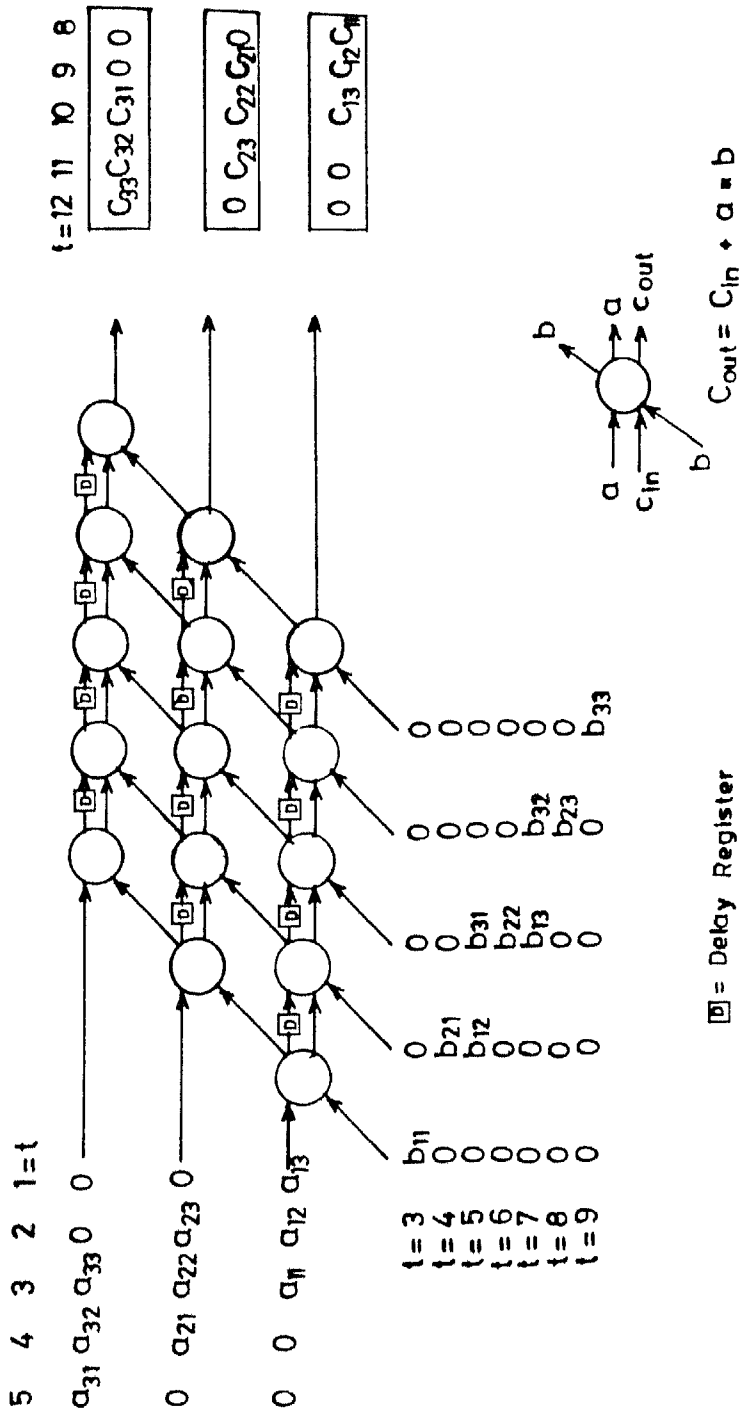


Fig 4.16 Systolic Solution for Matrix - Matrix Multiplication $C = A * B$ $p = (0\ 1\ -1)^t$

4 2 3 DISCRETE FOURIER TRANSFORM (DFT)

Let (x(n) n=0 1 N-1) be a finite length sequence The DFT of x(n) is defined as

$$y_k = \sum_{n=0}^{N-1} x_n * w^{nk} \qquad (4\text{-}6)$$

where $w = e^{-j2\pi / N}$ and k=0 1 N-1

Eq (4-6) can be represented in a matrix-vector multiplication format which is shown below for N=4

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (4\text{-}7)$$

The locally recursive version of eq(4-6) is given below

$$y_1^k = y_1^{k-1} + w_1^k * x_1^k$$
$$y_1^0 = 0 \quad , \quad y_1 = y_1^N$$
$$x_1^k = x_{1-1}^k \qquad x_1^k = x_{k-1} \qquad (4\text{-}8)$$
$$w_1^k = w^{(1-1)(k-1)}$$

The DG obtained from eq (4-8) is shown in figure 4-17 Since the DG contains only one type of cells the systolic solutions are homogeneous arrays Each cell does the multiply and accumulate operations (MAC) and hence is not indicated The index set and dependence vectors of the DG are given below

$$I^2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The first dependence vector is of variable x and the last is of the variable y There exists four valid projection vectors namely $(1\ 0)^t$ $(0\ 1)^t$ $(1\ 1)^t$ and $(1\ -1)^t$ and the resulting architectures are shown in figure 4-18 through 4-21 The schedule vector $(1\ 1)^t$ is used with the first three valid projection vectors and results in a computation time of (2*N-1)=7 clock cycles The schedule vector $(2\ 1)^t$ is used with the last projection vector and results in a computation time of ten clock cycles The results are listed in table 4-2

Table 4-2 Comparison of architectures for DFT (Matrix-vector multiplication algorithm)

Projection vector	Transformation matrix	Cells	Initial offset	computation time	Throughput
$(1\ 0)^t$	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	4	-	7	1
$(0\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	4	-	7	-
$(1\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	7	3	7	(1/2)
$(1\ -1)^t$	$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$	7	3	10	1

Solutions 1,2 and 4 have no final offset and are 'partitionable' Solution 3 has a final offset of three clock cycles and is not partitionable

For minumum chip area, the best solutions are 1 and 2 Note that the total required execution time includes the loading of

input sequence x for solution 1 and unloading of the output sequence y for the solution 2 Further all systolic solutions except 3 are partitionable providing the facility of mapping large DFTs onto these architectures

But efficient architectures can be obtained by employing the Horner's nested rule to the DFT algorithm Thus the DFT algorithm can be written as

$$y_i = x_0 + w^i(x_1 + w^i(x_2 + \quad + w^i(x_{N-2} + w^i(x_{N-1})))) \quad (4\text{-}9)$$

The above equation can be written as the following recursive equation which is localized

$$y_i^k = y_i^{k-1} * w_i^k + x_i^k$$

$$y_i^0 = 0 \quad , \quad y_i = y_i^N \quad (4\text{-}10)$$

$$x_i^k = x_{i-1}^k \quad , \quad x_1^k = x_{N-k}$$

$$w_i^k = w_i^{k-1} \quad , \quad w_i^i = w^{i-i}$$

The DG obtained from eq (4-10) is shown in figure 4-22 The DG is homogeneous consisting of MAC cells only The dependence vectors and index set of the DG are given below

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$I^2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix}$$

The first dependence vector is of variable x the next is of variable w and the last is of variable y There again exists four valid projection vectors The resulting architectures were

shown in figure 4-23 through 4-26 The transformation matrices and architectural details of these solutions are listed in table 4-3

Table 4-3 Comparison of architectures for DFT
(Horner s nested rule version)

Projection vectors	Transformation matrix	Cells	Initial offset	Computation time	Throughput
$(1\ 0)^t$	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	4	-	7	1
$(0\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	4	-	7	-
$(1\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	7	3	7	(1/2)
$(1\ -1)^t$	$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$	7	3	10	1

Solutions 1,2 and 4 have no final offset and are 'partitionable' Solution 3 has a final offset of three clock cycles and is not partitionable

Note that the architectures resulting from the DG in figure 4-22 have an advantage in that only boundary cells does the input/output operations where as in the architectures resulting from the DG of figure 4-17, all cells does the I/O operations

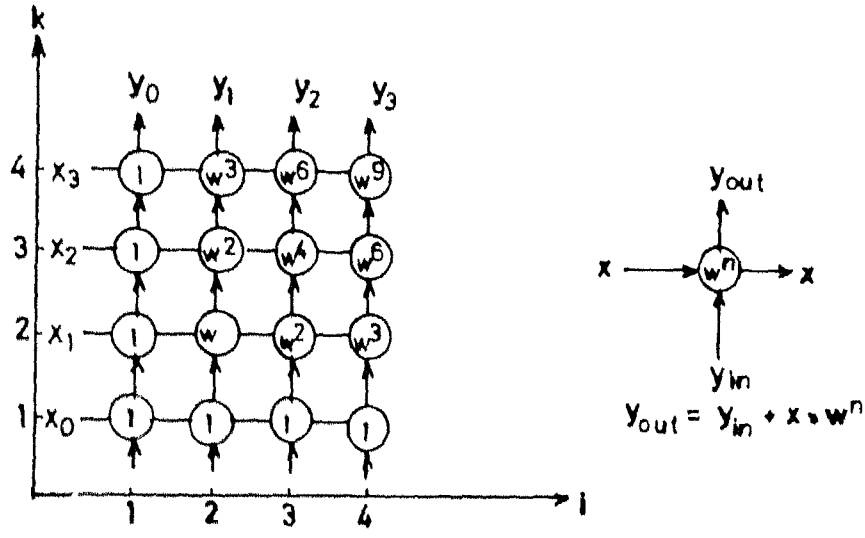


Fig 4.17 DG for DFT (Matrix - Vector Multiplication Version)

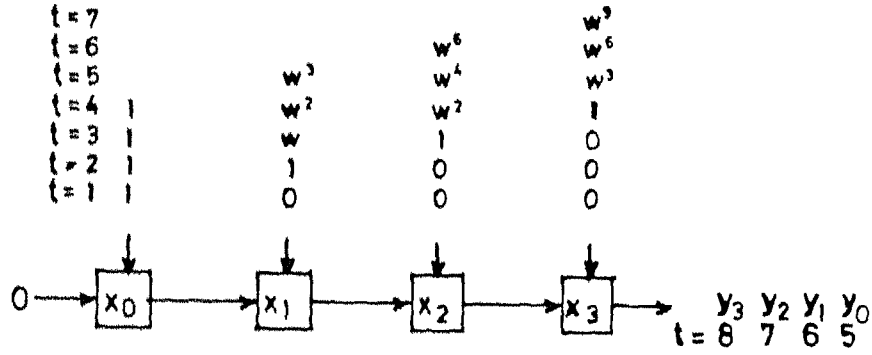


Fig 4.18 Systolic Solution for DFT , $p=(1\ 0)^t$

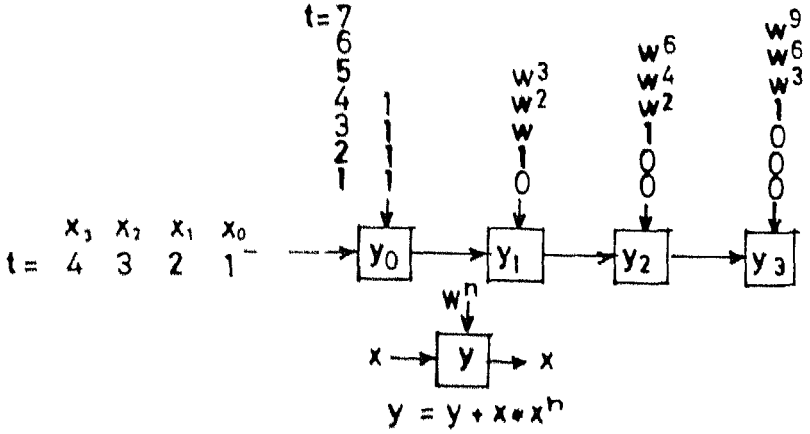


Fig 4.19 Systolic Solution for DFT $p=(01)^t$

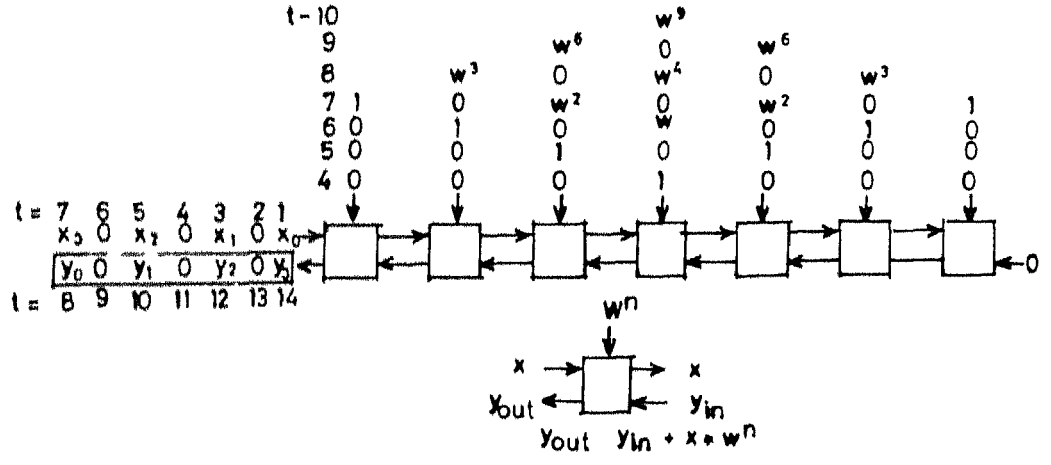
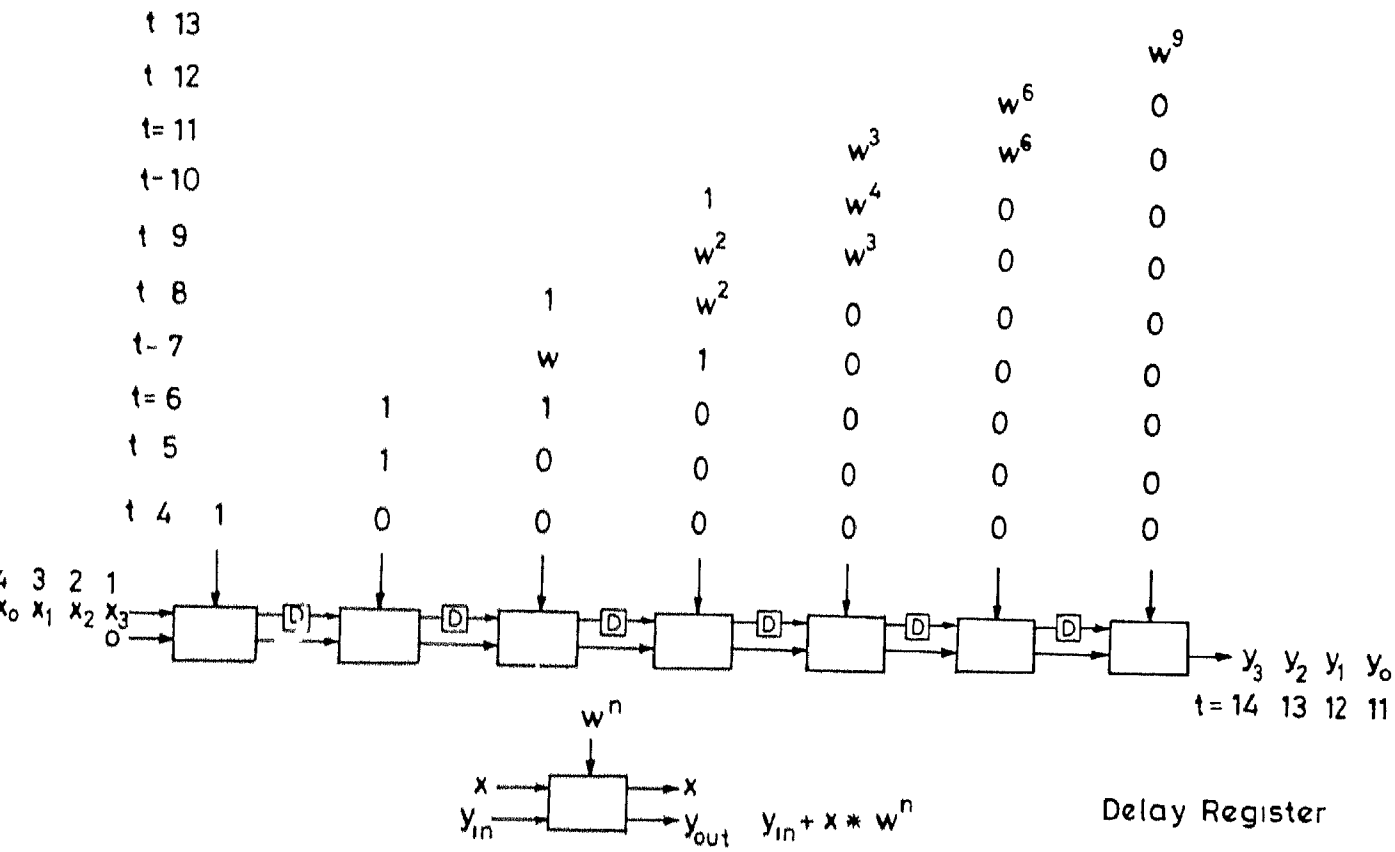


Fig 4.20 Systolic Solution for DFT $p=(11)^t$


t 13
t 12
t= 11
t-10
t 9
t 8
t- 7
t= 6
t 5
t 4
4 3 2 1
x_0 x_1 x_2 x_3
0
1
1
0
1
w
1
1
0
0
1
w^2
w^2
1
0
0
0
w^3
w^4
w^3
0
0
0
0
w^6
w^6
0
0
0
0
w^9
0
0
0
0

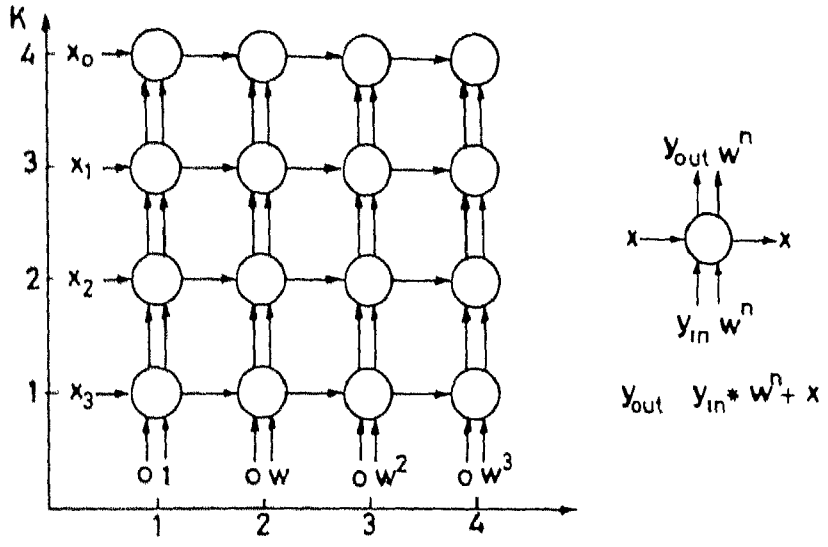


Fig 4 22 DG for DFT (Horner's nested rule version)

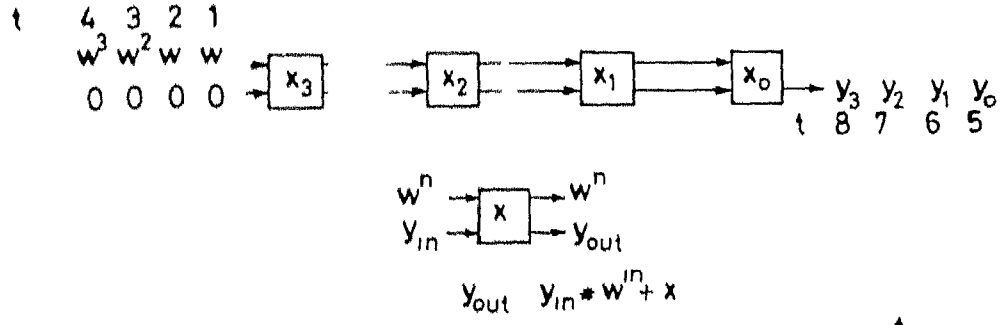


Fig 4 23 Systolic solution for DFT , p $(1\ 0)^t$

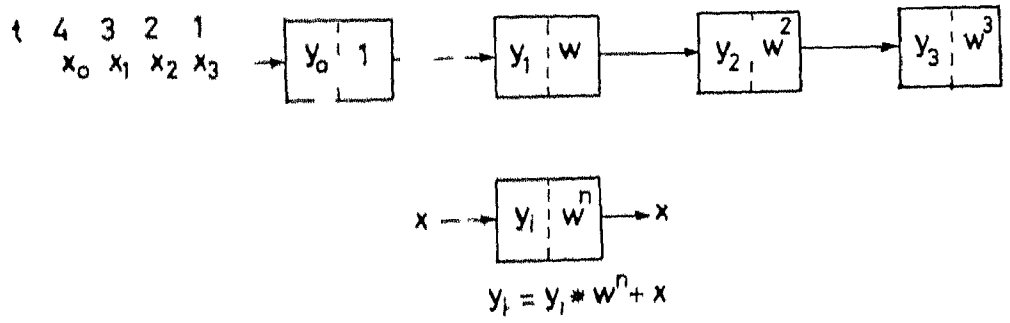


Fig 4-24 Systolic solution for DFT , p $(0\ 1)^t$

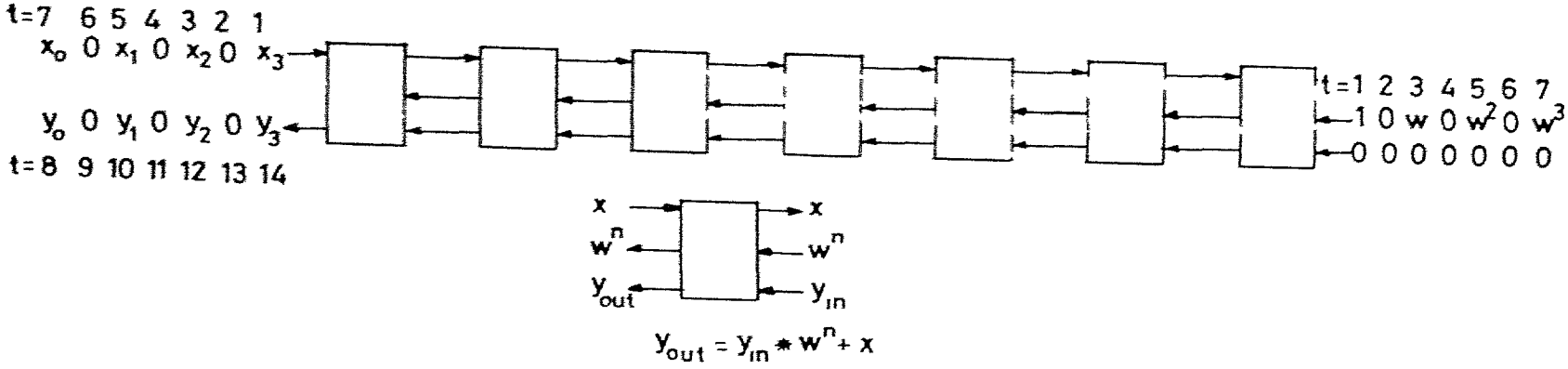


Fig 4-25 Systolic solution for DFT $p=(1\ \ 1)^t$

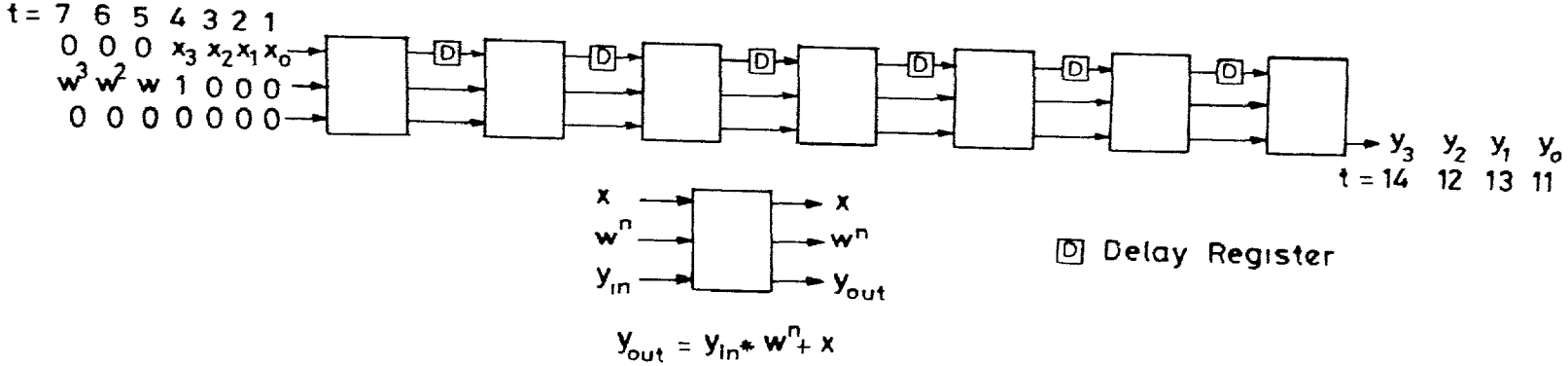


Fig 4 26 Systolic solution for DFT $p_-(1\ -1)^t$

4 2 4 LU DECOMPOSITION

An NXN LU decomposition algorithm, C = A*B where C = (c_{ij}) is an NXN non-singular matrix, A = (a_{ij}) is an NXN lower triangular matrix with a unit diagonal and B = (b_{ij}) is an upper triangular matrix, can be described as follows

$$a_{ij} = (c_{ij} - \sum_{k<j} a_{ik} * b_{kj})/b_{jj} \quad \text{for } i=2\ 3 \quad N \text{ and } j=1\ 2 \quad i-1$$

$$b_{ij} = c_{ij} - \sum_{k<i} a_{ik} * b_{kj} \quad \text{for } i=1\ 2 \quad N \text{ and } j=1 \quad N \qquad (4\text{-}11)$$

For N=4, the matrices are as follows

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_{21} & 1 & 0 & 0 \\ a_{31} & a_{32} & 1 & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ 0 & b_{22} & b_{23} & b_{24} \\ 0 & 0 & b_{33} & b_{34} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

The eq (4-11) can be formulated as

$$c_{ij}^{k} = c_{ij}^{k-1} - a_{i}^{k} * b_{j}^{k}$$
$$a_{i}^{k} = c_{ik}^{k-1} / c_{kk}^{k-1} \quad , \quad a_{i}^{k} = a_{ik} \qquad (4\text{-}12)$$
$$b_{j}^{k} = c_{kj}^{k-1} \quad , \quad b_{j}^{k} = b_{kj}$$
$$\text{for } k=1,2, \ldots ,N \quad , \quad k < i < N \text{ and } k < j < N$$

The algorithm of the eq (4-12) needs to be localized as there are missing indices of variables a and b The locally recursive version of the eq (4-12) is given below

$$c_{i\ j}^{k} = c_{i,j}^{k-1} - a_{i\ j}^{k} * b_{i,j}^{k}$$
$$a_{i,j}^{k} = c_{i,j}^{k-1} / b_{i,j}^{k} \quad \text{if } j = k$$
$$= a_{i,j-1}^{k} \quad \text{if } j \neq k \qquad (4\text{-}13)$$

$$b_{i,j}^{k} = c_{i,j}^{k-1} \quad \text{if } i = k$$

$$= b_{i-1,j}^{k} \quad \text{if } i \neq k$$

$c_{i,j}^{0} = c_{i,j}$, $a_{i,j} = a_{i,N}^{j}$, $b_{i,j} = b_{N,j}^{i}$

for $k = 1,2 \ldots N$, $i = k, \ldots ,N$ and $j = k, \ldots N$

The localized DG is shown in figure 4-27 Note that the dependence lines in the k-direction that run from the point (i j,k) to (i,j,k+i) are not drawn

The DG is non-homogeneous consisting of 'multiply and subtract operation (MS) cells, 'division operation' (D) cells and 'equating operation' (E) cells Let us denote the DG consisting of MS cells as DG-1, DG consisting of D cells as DG-2 and the DG consisting of E cells as DG-3 The dependence vectors of the various DGs are given below

$$D_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad D_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The first dependence vector of D_1 is due to the pipelining of variable a, the second vector is due to the pipelining of the variable b and the last vector is of variable c DG-2 has only one dependence vector to pipeline the variable b DG-3 has no dependence vector Further the following dependence vectors exists between the DGs

$$D_{12} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad , \quad D_{13} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad , \quad D_{21} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad , \quad D_{31} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$D_{32} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and no dependence vector exists from DG-2 to DG-3

The index set of various DGs are given below

$$I_1^3 = \begin{bmatrix} 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4 & 3 & 3 & 4 & 4 & 4 \\ 2 & 3 & 4 & 2 & 3 & 4 & 2 & 3 & 4 & 3 & 4 & 3 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 \end{bmatrix}$$

$$I_2^3 = \begin{bmatrix} 2 & 3 & 4 & 3 & 4 & 4 \\ 1 & 1 & 1 & 2 & 2 & 3 \\ 1 & 1 & 1 & 2 & 2 & 3 \end{bmatrix}$$

$$I_3^3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \\ 1 & 2 & 3 & 4 & 2 & 3 & 4 & 3 & 4 & 4 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \end{bmatrix}$$

It has been found that there exists only one valid projection vector, namely $(1\ 1\ 1)^t$ and the schedule vector $(1\ 1\ 1)^t$ results in a minimum computation time of ten clock cycles The processor basis A corresponding to this projection vector is $\begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$ Hence the transformation matrix T is $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$ The transformed index set and dependencies are printed below

$$T*I_1^3 = \begin{bmatrix} 5 & 6 & 7 & 6 & 7 & 8 & 7 & 8 & 9 & 8 & 9 & 9 & 10 & 11 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 1 & 1 & 2 & 2 & 1 \\ 0 & -1 & -2 & 1 & 0 & -1 & 2 & 1 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$$

$$T * I_2^3 = \begin{bmatrix} 4 & 5 & 6 & 7 & 8 & 10 \\ 1 & 2 & 3 & 1 & 2 & 1 \\ 1 & 2 & 3 & 1 & 2 & 1 \end{bmatrix}$$

$$T * I_3^3 = \begin{bmatrix} 3 & 4 & 5 & 6 & 6 & 7 & 8 & 9 & 10 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -2 & -3 & 0 & -1 & -2 & 0 & -1 & 0 \end{bmatrix}$$

$$T * D_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{bmatrix} \quad , \quad T * D_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$T * D_{12} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \quad , \quad T * D_{13} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \quad , \quad T * D_{21} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

$$T * D_{31} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad , \quad T * D_{32} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The architecture built with this information is shown in figure 4-28 Note that the transformed dependencies $T * D_i$ indicates the links that exists within SSA-i where as $T * D_{ij}$ indicates the links that exists from SSA-i to SSA-j Further note that a latch has been used in the place of E cell The architecture of figure 4-28 has an initial offset of three clock cycles and final offset of three clock cycles Hence the total required execution time is (3+3+10)=16 clock cycles The throughput per pipeline is (1/3) as only one output component appears for every three clock cycles in each pipeline

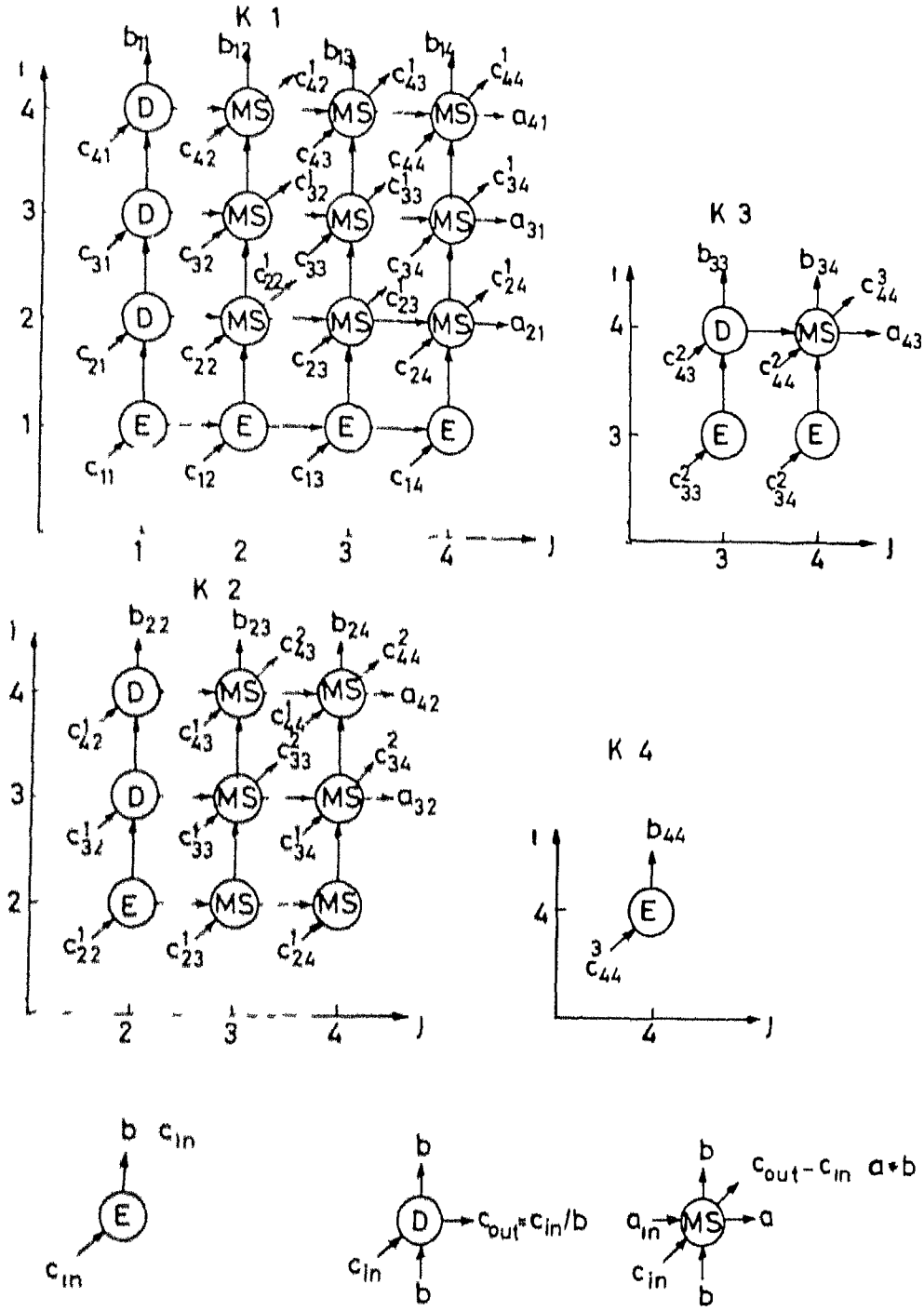


Fig 4 27 DG for LU decomposition c = AB

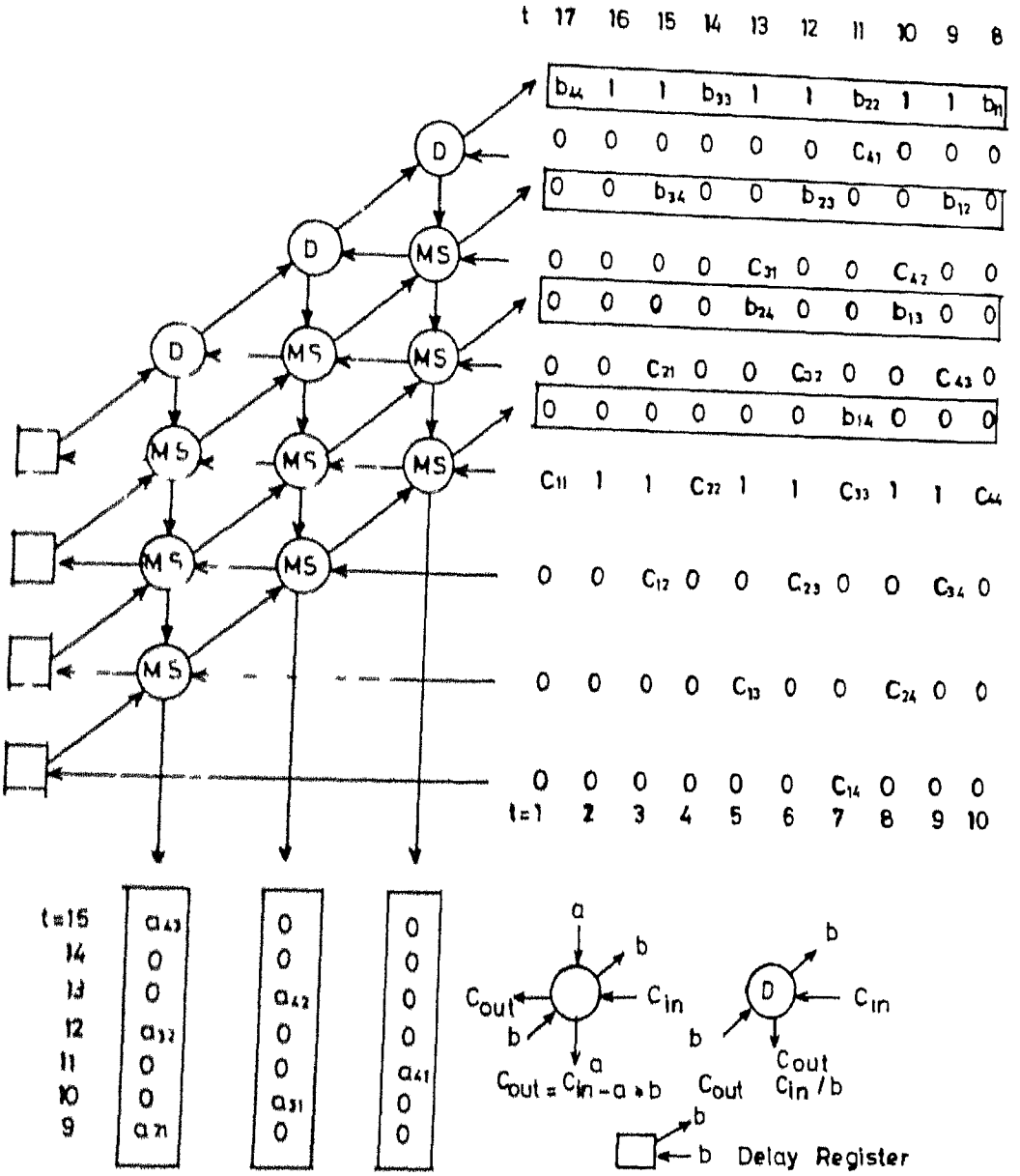


Fig 4.28 Systolic Solution for LU Decomposition $C = A\ B$ $p=(111)^t$

4 2 5 FINITE IMPULSE RESPONSE (FIR) FILTER

A common representation of a digital filter of order N is given by

$$y_n = \sum_{i=1}^{N} r_i * y_{n-i} + \sum_{i=0}^{M} w_i * x_{n-i} \qquad (4\text{-}14)$$

where x_n and y_n are input and output signals respectively

There are two classes of basic digital filters If the coefficients r_i in eq (4-14) are all zero, then the filter is known as moving average' (MA) filter else it is known as 'autoregressive moving average' (ARMA) filter Note that MA filters are FIR filters and ARMA filters are IIR filters

Thus for FIR filter, the input-output can be described as

$$y_n = \sum_{i=0}^{M} w_i * x_{n-i} \qquad (4\text{-}15)$$

The eq (4-15) can be represented by the following backward recurrence equation

$$\begin{aligned} y_i^j &= y_i^{j-1} + w_i^j * x_i^j \\ w_i^j &= w_{i-1}^j \quad , \quad w_{i-1} = w_1^i \\ x_i^j &= x_{i-1}^{j-1} \quad , \quad x_{i-1} = x_i^i \\ y_i^0 &= 0 \quad , \quad y_{i-1} = y_i^{M+1} \end{aligned} \qquad (4\text{-}16)$$

The DG obtained from eq (4-16) is shown in figure 4-29 for M = 4 All the cells does the MAC operation and hence the systolic solutions are homogeneous arrays Note the index set and dependence vectors of the DG It has been found that there

exists four valid projection vectors and the architectures are shown in figure 4-30 through 4-33 The results are furnished in table 4-4 These architectures can be constructed by computing the transformed index sets and dependencies and following the procedures outlined in the previous sub-sections Note that the pipeline of variable x in figure 4-34 exhibits non-near neighbour communication This possibility can be ruled out by putting the condition $T * D_i \neq m * T * D_j$ where D_i and D_j are dependence vectors of a homogeneous DG and m is a constant greater than one This is not incorporated in the ADSA program because the meaning of the term 'local connectivity' is quite ambiguous

Table 4-4 Comparison of architectures for FIR filter

Projection vector	Transformation matrix	Cells	Initial offset	Computation time	Throughput
$(1\ 0)^t$	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	5	-	9	1
$(0\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	5	4	9	-
$(1\ 1)^t$	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	5	4	9	(1/2)
$(1\ -1)^t$	$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$	9	4	13	1

All architectures except that of figure 4-32 are 'partitionable'

Further a different set of architectures which are not reported here can be obtained by the following forward recurrence equations

$$y_i^j = y_i^{j+1} + w_i^j * x_i^j$$

$$w_i^j = w_{i-1}^j \qquad w_{i-1} = w_i^1 \qquad (4\text{-}17)$$

$$x_i^j = x_{i-1}^{j-1} \qquad x_{i-1} = x_i^1$$

$$y_i^{M+1} = 0 \quad , \qquad y_{i-1} = y_i^0$$

In many applications it is desirable to design filters that have linear phase In this way signals in the passband of the filter are reproduced exactly at the filter output except for a time delay corresponding to the slope of the phase For a linear phase FIR filter

$$w_n = w_{M-n} \quad \text{for} \quad n = 0,1, \ldots ,M-1$$

Hence the eq (4-15) gets modified as follows

$$y_n = \sum_{i=0}^{S} w_i * x_{n-i} + \sum_{i=0}^{S-1} w_i * x_{n-M+i} \qquad (4\text{-}18)$$

where $S = M/2$

Let the two summation terms in the above equation be represented by P and R respectively Then the eq (4-18) can be represented by the following backward recurrence equations

$$P_i^j = P_i^{j-1} + w_i^j * x_i^j \qquad \text{for } j = 0,1, \ldots M$$

$$R_i^j = R_{i-2}^{j-1} + w_i^j * x_i^j \qquad \text{for } j = 0,1, \ldots ,M-1$$

$$y_i^j = P_i^{j-1} + R_{i-2}^{j-1} + w_i^j * x_i^j \qquad \text{for } j = M+1 \qquad (4\text{-}19)$$

$$w_i^j = w_{i-1}^j \quad , \qquad w_i^j = w_{j-1}$$

$$x_i^j = x_{i-1}^{j-1} \qquad x_i^1 = x_{i-1}$$

$$P_i^0 = 0 \qquad R_i^0 = 0 \qquad , \qquad y_{i-1} = y_i^{M+1}$$

The DG constructed from eq (4-19) is shown in figure 4-34 for M = 4 There exists only one valid projection vector namely $(1\ 0)^t$ and the schedule vector $(1\ 1)^t$ is used Thus the transformation matrix T is $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ The resulting architecture is shown in figure 4-35 This architecture has no initial and final offsets Its computation time and hence total required execution time is seven clock cycles and throughput is one Note that this architecture requires only half the number of cells as compared with the architecture of figure 4-30, but there is an increase in the complexity of the cell It is possible to design an architecture with less complex cell structure by remodelling the DG in such a way that the 'multiplication of w and x variables occurs after the 'addition of proper x components [18]

Further note that the architecture for even number of filter coefficients will be different For example, for M = 5, the architecture will have three MAC-1 cells followed by an 'adder cell

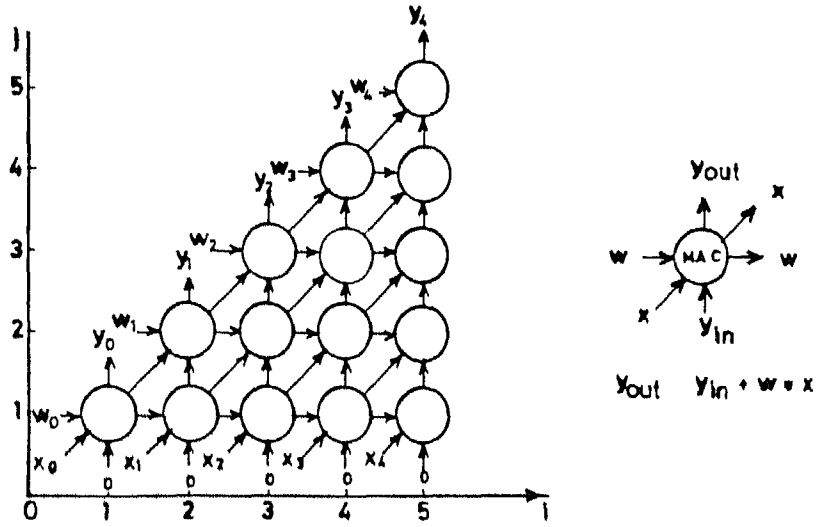


Fig 4.29 DG for FIR Filter

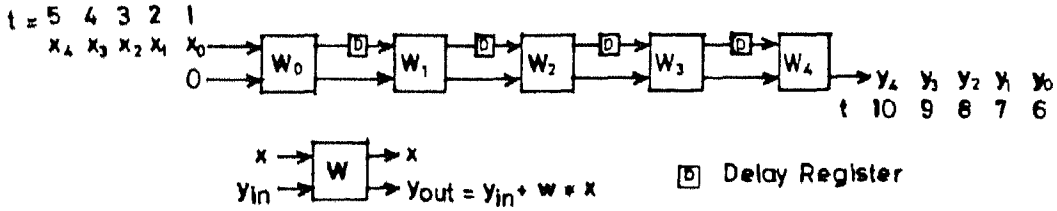


Fig 4.30 Systolic Solution for FIR Filter $p=(1\ 0)^t$

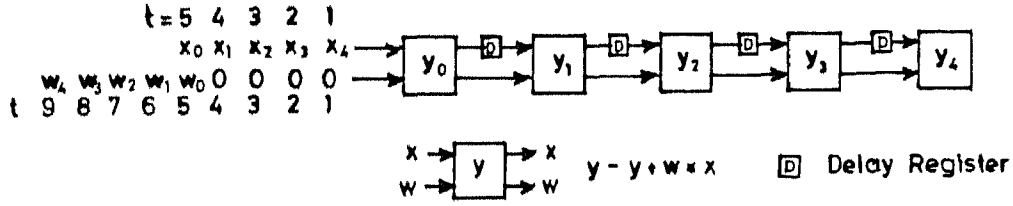


Fig 4.31 Systolic Solution for FIR Filter $p=(0\ 1)^t$

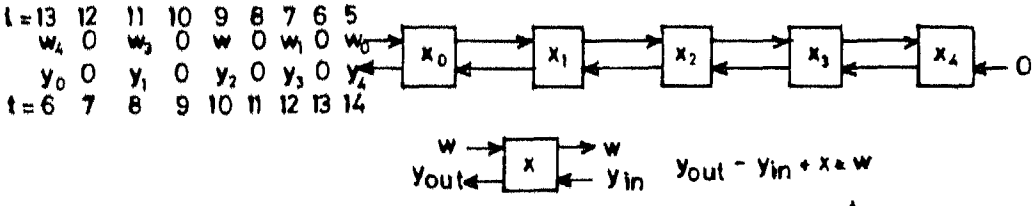
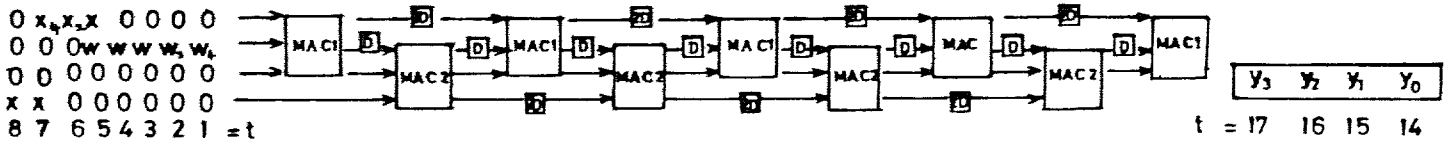


Fig 4.32 Systolic Solution for FIR Filter $p=(1\ 1)^t$



$y_{out} = y_{in} \quad x * w$ $\qquad$ $y_{out} = y_{in} * x \quad w$

D = Delay Register

FD = Delay Register for Two Clock Cycles

Fig 4 33 Systolic Solution for FIR Filter $p = (1 \ -1)^t$

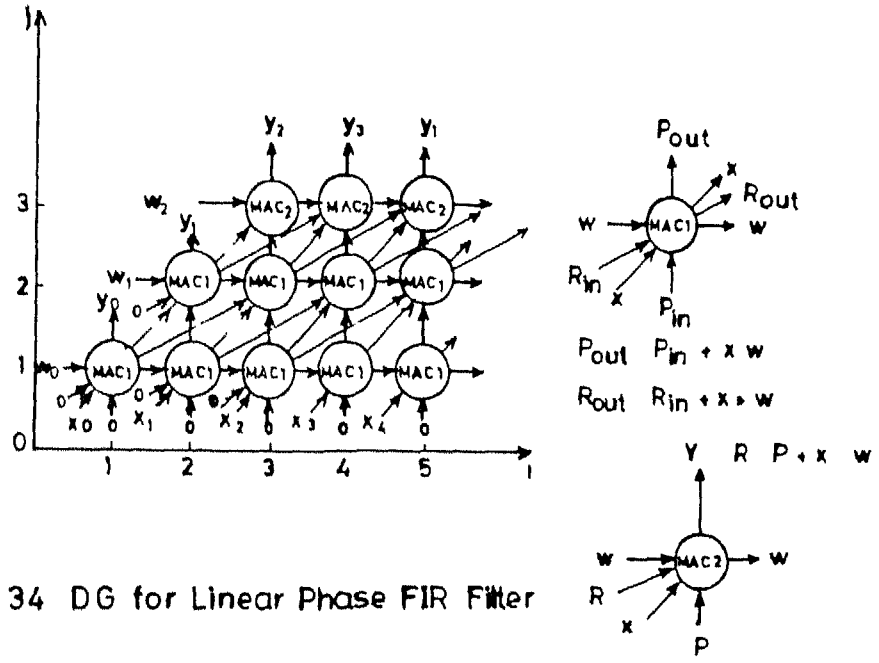


Fig 4.34 DG for Linear Phase FIR Filter

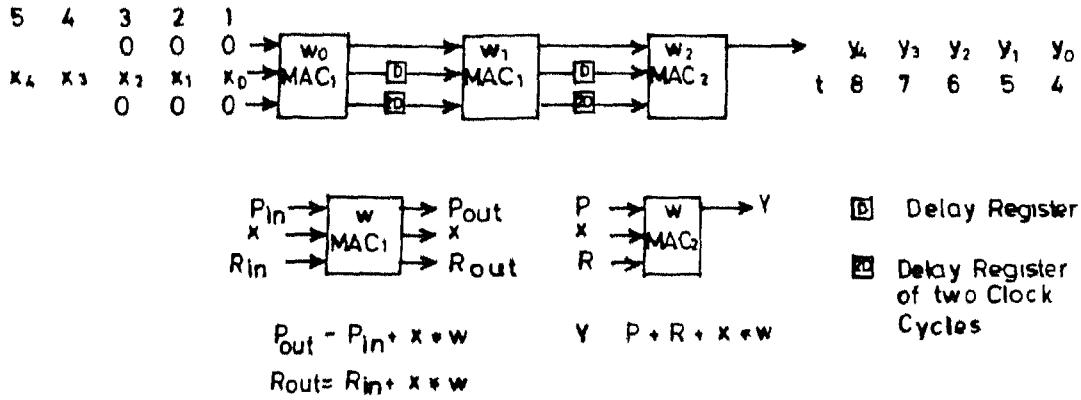


Fig 4.35 Systolic Solution for Linear Phase FIR Filter $p=(1\ 0)^t$

4.2.6 RECURSIVE CONVOLUTION

The recursive convolution equation can be expressed as

$$y_n = \sum_{i=1}^{N} r_i * y_{n-i} \qquad (4\text{-}20)$$

Eq.(4-20) can be represented by the following forward recurrence equation .

$$y_i^{\,j} = y_i^{\,j+1} + r_i^{\,j} * R_i^{\,j}$$

$$R_i^{\,0} = y_i^{\,1} \quad ; \quad R_i^{\,j} = R_{i-1}^{\,j-1}$$

$$r_i^{\,j} = r_{i-1}^{\,j} \quad ; \quad r_1^{\,j} = r_{j-1} \qquad (4\text{-}21)$$

$$y_1^{\,N+1} = y_0 \quad ; \quad y_i^{\,N+1} = 0 \quad ; \quad y_i = R_{i+N+1}^{\,N+1}$$

for $j = N, N-1, \ldots 2, 1$

The DG corresponding to the eq.(4-21) is shown in figure 4-36 for N = 4. The systolic solution will be a non-homogeneous array because the DG contains MAC cells and E cells. There exists only one valid projection vector, namely $(1\ 0)^t$ and a schedule vector $(2\ -1)^t$ is used. Thus the transformation matrix is $\begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix}$. The resulting architecture which is obtained from the transformed index sets and dependencies is shown in figure 4-37. Note that a 'latch' can be used in the place of E cell. The architecture has an initial offset of N (four) clock cycles and final offset of N (four) clock cycles. The computation time is 2*m-1 (eleven) clock cycles where m is the length of the output sequence. And its *throughput* is half.

An architecture with a throughput of one can be obtained by applying the *divide-and-conquer* technique. This modified array has about the same area as the array of figure 4-37.

From eq.(4-20), we obtain

$$y_{n-1} = \sum_{i=1}^{N} r_i * y_{n-1-i} \qquad (4\text{-}22)$$

Further eq.(4-20) can be written as

$$y_n = r_1 * y_{n-1} + \sum_{i=2}^{N} r_i * y_{n-i} \qquad (4\text{-}23)$$

Hence substituting eq.(4-22) into eq.(4-23), we have

$$y_n = r_1 * \sum_{i=1}^{N} r_i * y_{n-1-i} + \sum_{i=2}^{N} r_i * y_{n-i}$$

$$= \sum_{i=2}^{N+1} a_i * y_{n-i} \qquad (4\text{-}24)$$

where $a_i = r_i + r_1 * r_{i-1}$ for $2 \le i \le N$ and $a_{N+1} = r_1 * r_N$

This modification does not change the total number of coefficients.

Dividing the eq.(4-24) into sub-equations, we have

$$P_n = \sum_{m=1}^{N/2} a_{2m} * y_{n-2m}$$

$$Q_n = \sum_{m=1}^{N/2} a_{2m+1} * y_{n-(2m+1)} \qquad (4\text{-}25)$$

In eq.(4-25), it is assumed that N is even without loss of generality. The two sub-equations of eq.(4-25) can be treated as two seperate problems and hence two seperate DGs can be obtained. Further these two DGs are linked by the relation $y_n = P_n + Q_n$. Thus the combined DG which is obtained from the following recursion equations is shown in figure 4-38 for N = 4.

$$P_i^{\,j} = P_i^{\,j-1} + a_i^{\,j} * R_i^{\,j}$$

$$a_i^{\,j} = a_{i-1}^{\,j} \qquad a_{N+2-2j} = a_1^{\,j}$$

$$R_i^j = R_{i-2}^{j+1} \quad , \quad R_i^{N/2+1} = y_i^{N/2+1} \quad \text{and} \quad P_i^0 = 0$$

for $j = 1, 2, \ldots, N/2$

$$Q_i^j = Q_{i-2}^{j+1} + a_i^j * S_i^j \qquad (4\text{-}26)$$

$$a_i^j = a_{i-1}^j \qquad a_{2j-(N+1)} = a_i^j$$

$$S_i^j = S_i^{j-1} \quad , \quad S_i^{N/2+1} = y_i^{N/2+1} \quad \text{and} \quad Q_0^j = 0$$

for $j = N/2+2, \ldots, N+1$

$$y_i^{N/2+1} = P_i^{N/2} + Q_{i-2}^{N/2+2}$$

$$S_i^{N/2+2} = S_{i-1}^{N/2+1}$$

Note that we have introduced two variables R and S to propagate the components of y

let us denote the DG consisting of MAC-1 cells alone by DG-1 the DG consisting of 'adder' (A) cells alone by DG-2 and the DG consisting of MAC-2 cells alone by DG-3 Note the index sets and the dependence vectors within a DG and from one DG to another DG It has been found that there exists only one valid projection vector, namely $(1\ 0)^t$ and a schedule vector $(1\ 1)^t$ is used Hence the geometric transformation matrix T is $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ The resulting architecture is shown in figure 4-39 It has an initial offset of N/2 (two) clock cycles and a final offset of N/2 (two) clock cycles The computation time is l clock cycles where l is the length of the input sequence

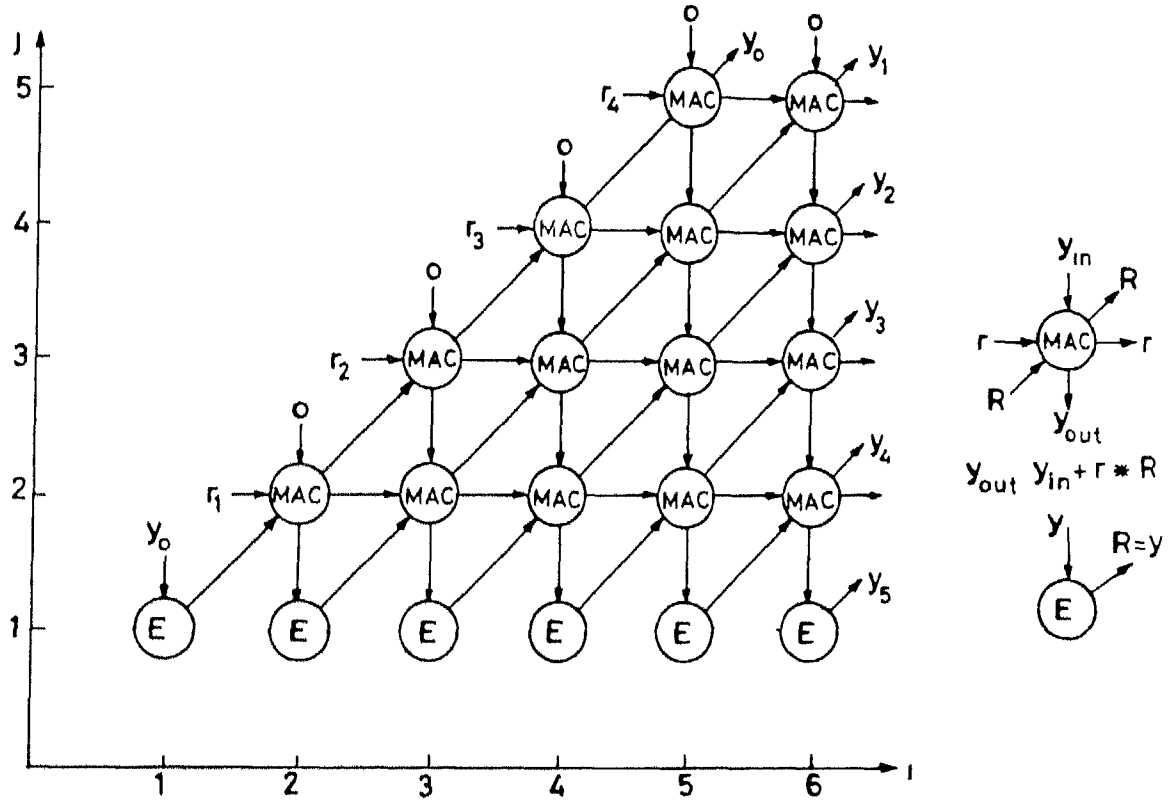


Fig 4 36 DG for recursive convolution

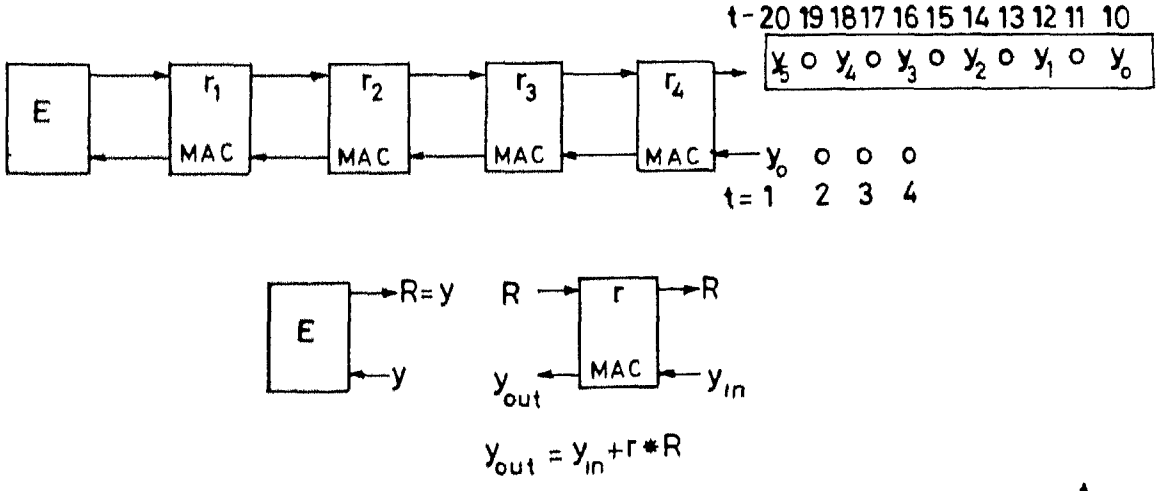


Fig 4 37 Systolic solution for recursive convolution, $p-(1\ 0)^t$

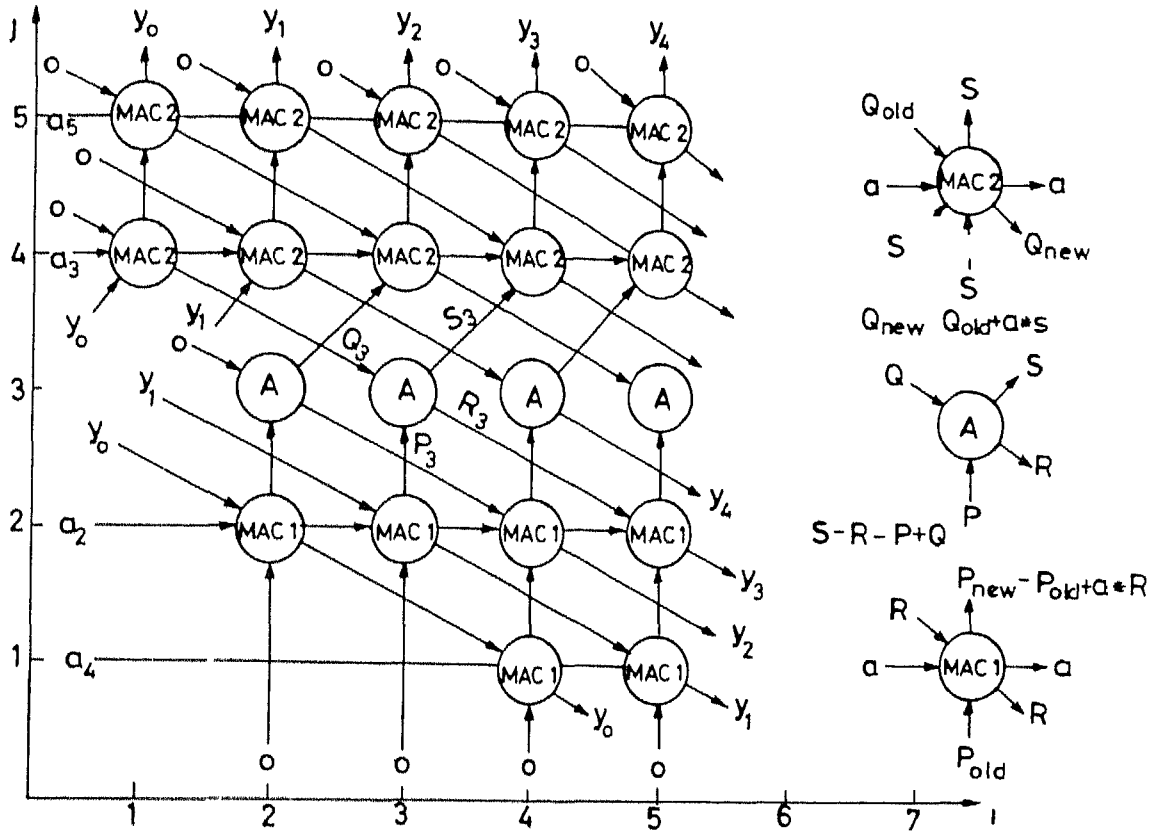
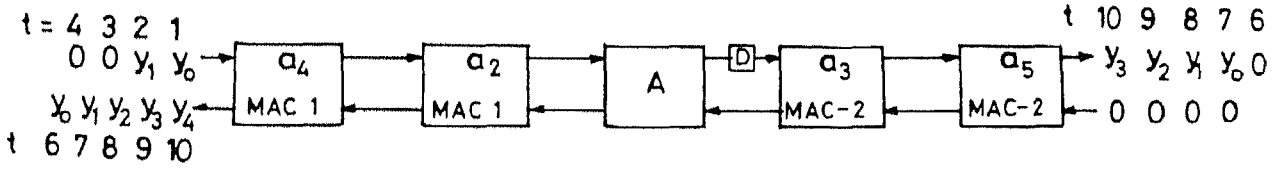


Fig 4 38 DG for recursive convolution (Divide and conquer version)



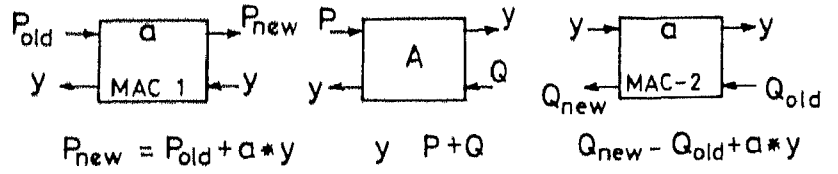


Fig 4 39 Systolic solution for recursive convolution , $P-(1\ 0)^t$
(Divide and conquer version)

4 2 7 INFINITE IMPULSE RESPONSE (IIR) FILTER

For an IIR filter the input-output can be described as

$$y_n = \sum_{i=0}^{M} w_i * x_{n-i} + \sum_{i=1}^{N} r_i * y_{n-i} \qquad (4\text{-}27)$$

A set of architectures which are not reported here can be obtained from a DG which is obtained by combining the DGs of a FIR filter and a recursive convolution equation corresponding to the first and second summation terms in the eq (4-27) respectively

However a new architecture which is obtained by remodelling the eq (4-27) is presented here

The eq (4-27) can be written as

$$y_n = \sum_{i=0}^{n} c_i * x_{n-i}$$

$$\text{where} \quad c_i = \sum_{j=0}^{M} w_j * r_{i-j}^{(i)} \qquad (4\text{-}28)$$

$$\text{and} \quad r_j^{(i)} = \sum_{m=1}^{N} r_m * r_{j-m}^{(i)}$$

The above recurrence equations are obtained by first expressing the past output samples in terms of the input x_n only (starting with $y_0=w_0*x_0$) and then substituting them in the expression for the present output sample so as to make it to depend upon the input x_n only

In the eq (4-28), the first two recurrence equations corresponds to the 'linear convolution' (or FIR filter) algorithm and the last recurrence equation corresponds to the

recursive convolution algorithm The resulting architecture is shown in figure 4-40 for M = 2 N = 3 and an input sequence of five samples This architecture is obtained by concatenating the architectures of recursive convolution with the architectures for FIR filter Note that many architectures for IIR filter can be obtained by choosing the different architectures for recursive convolution and FIR filter algorithms In the architecture of figure 4-40 we have used the architecture of figure 4-39 for recursive convolution (note that the coefficients r_1 gets modified to a_1 as discussed in sub-section 4 2 6) and the architectures of figure 4-30 and figure 4-31 for FIR filter algorithm

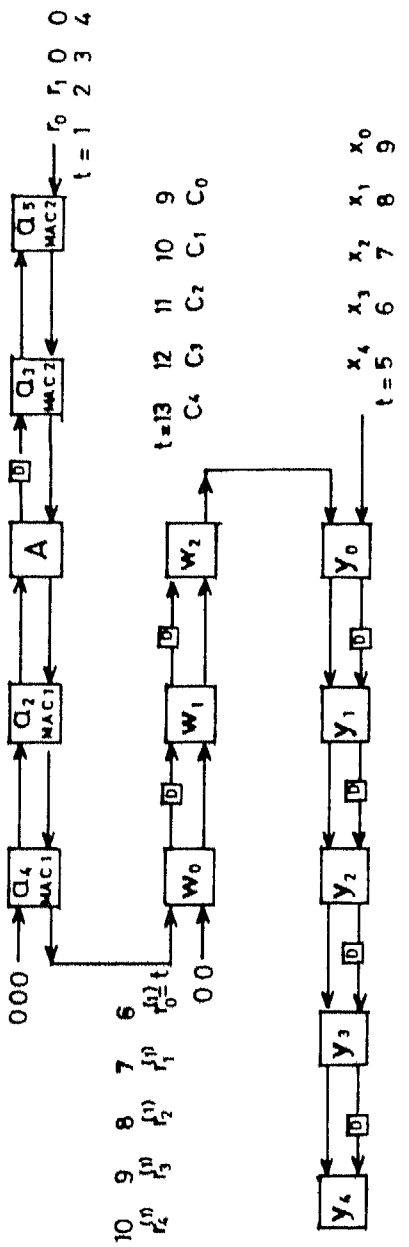


Fig 4.40 Systolic Solution for IIR Filter

CHAPTER 5

CONCLUSIONS AND SCOPE FOR FURTHER WORK

Systolic array structures are known for their regular structures and simple cells These arrays operate *synchronously* performing dependent computations through the pipelines and independent computations in parallel The advantage of using extensive *pipelining and multiprocessing* makes them suitable for the implementation of many computationally intensive problems Although they have many advantages in their architectural properties, their design involves stringent data scheduling the *right data has to reach the right cells at the right time* in order to preserve algorithm s correctness This involves the determination of the spatial and timing distributions of data

Several attempts have been made to systematize the design of systolic arrays either directly from an algorithm or from an intuitive non-systolic design Presented here is an introduction of a *simple and systematic systolic array design method aimed at solving some of the existing design problems* The design method in general, follows three major steps the *algorithm representation algorithm model and architecture specification* The *algorithm representation* involves transforming the algorithm in the form of mathematical expressions, into a set of locally recursive equations At present, this process is ad hoc but research is going on to systematize this step In *algorithm model* step a

Dependence Graph (DG) is systematically obtained from these recursive equations From this model the computations are first scheduled and then grouped among a set of cells such that the systolic array properties such as simple and regular data and control communication, which are represented in terms of a set of conditions based on the dependence information and index set of the DG, are preserved The grouping of computations is done via geometric projections Thus the valid projection vectors are systematically derived from the DG In *architecture specification* step, processor basis A corresponding to each valid projection vector is determined as the graph-based projection procedure may be described in terms of algebraic transformations Then a geometric transformation matrix $T = \begin{bmatrix} s^t \\ A^t \end{bmatrix}$ is formed The design information such as the number of cells operations performed in each cell and data timings are extracted from the transformed index set of the DG Cell interconnections and data movement are extracted from the transformed dependencies

Since the state-of-art of current technology does not recommend three or higher dimensional architectures the algorithm for *multiprojection* scheme which maps four or higher dimensional DGs such as *2D convolution* onto a *planar* systolic array, is developed Further this method supports the scheme of *mapping problem of larger size onto the resulting smaller architecture,* which is not so with many of the existing methods

As the scope of further work is concerned the concepts of *multiprojection* and *partitioning* can be readily incorporated into the ADSA package More research should be oriented towards selecting a best algorithm for the problem under consideration as enumerated for FIR filter, DFT and recursive convolution Further the step of transforming the algorithm into locally recursive equations should be made automatic A new optimization theory is needed to be developed which not only considers the area time and I/O bandwidth measures (note that the architectures with non-stationary data needs more I/O bandwidth) but also the purpose for which the architecture will be used etc etc The *non linear transformations* such as $\begin{bmatrix} \lceil \frac{1}{2} \rceil & 1 \\ \text{mod2} & 1 \end{bmatrix}$ of Quinton [9] which produces *block convolver* (an architecture for linear convolution, whose throughput can made as large as possible, of course with corresponding increasemthe number of cells) and the cell sharing transformation of Cappello [5] are quite interesting A comparative study of the method presented in this thesis, Moldovan's method [12], Cappello's method [5] and Quinton's method [9] is worth rendering

APPENDIX

A 1 PARTITIONING

Consider a FIR filter algorithm described by

$$y_m = \sum_{i=0}^{M} c_i * x_{m-i} \qquad (A-1)$$

The DG which is obtained from the backward recurrences of eq (4-16) is shown in figure A-1 for M=7 and m=7 As already mentioned in section 4 2 5 there exists three valid projection vectors, namely $(1\ 0)^t$ $(0\ 1)^t$ and $(1\ 1)^t$ for the schedule vector $(1\ 1)^t$ The projection vector $(0\ 1)^t$ has been considered for further discussion Thus, the transformation matrix $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ The resulting architecture which is obtained from the transformed index set and dependencies is shown in figure A-2 Note that this architecture needs as many cells as the number of samples of the output sequence y_m which is not feasible Thus, we need to map the computations of a larger size problem onto the smaller architecture and this is known as the *partitioning'* It is a basic requirement in many practical system designs as can be recalled by Heller s tripartite corollary [20] to Murphy's law

No matter what special purpose device is available there is a problem too large for it

The problem will manifest itself only after the device is acquired and can no longer be modified

The problem can not be ignored

To design the partitioning scheme, the following factors should be taken into account [10]

1 Minimum overall computation time

2 Minimum control overheads

3 Balanced tradeoff between external communication and local memory

It is necessary to specify *at what time and in which* processor the computation of each index point of the DG takes place For a systematic mapping from the DG onto a systolic array the DG is regularly partitioned into many blocks, each consisting of a cluster of nodes in the DG For convenience of presentation the following mathematical notations are adopted Suppose that an n-dimensional DG is projected to an (n-1)-dimensional array of size $L_1 X L_2 X \ldots X L_{n-1}$ The array is partitioned into $M_1 X M_2 X \ldots X M_{n-1}$ blocks, where each block is of size $Z_1 X Z_2 X \ldots X Z_{n-1}$ Obviously, for i=1 2, n-1 $Z_i = L_i / M_i$

There are two methods for mapping the partitioned DG to an array locally sequential globally parallel (LSGP) method and locally parallel globally sequential (LPGS) method

1 LSGP scheme

In the LSGP scheme, one block is mapped to one processor Thus the number of blocks is equal to the number of processors in the array, i e , the array size equals to the product $M_1 X M_2 X \ldots X M_{n-1}$ Each processor sequentially executes the nodes (indices) of the corresponding block In order to store

the node data in the block local memory within each processor is needed, i e local memory size should be $cXZ_1XZ_2X \quad XZ_{n-1}$ where c is a small constant (usually 1 2 or 3) depending on the data dependencies As long as the local memory is large enough for the computation under consideration, the LSGP approach is quite appealing

2 LPGS scheme

In the LPGS scheme the block size is chosen to match the array size, i e one block can be mapped to one array All nodes within one block are processed concurrently i e *locally parallel* One block after another block of node data are loaded into the array and processed in a sequential manner i e , *globally sequential* In this scheme, local memory size in the processor can be kept constant, independent of the size of computation All intermediate data can be stored in certain buffers outside the processor array Usually simple First-in-First out (FIFO) buffers are adequate for storing and recirculating the intermediate data efficiently [18]

The LPGS scheme [12] can be readily incorporated into the ADSA package and hence will be discussed in detail

Suppose that the algorithm of the figure A-1 has to mapped onto a linear array of fixed size, say P=4 Consider the transformation matrix $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ again which has been found for the DG under consideration This transformation can be used for

partitioning the algorithm if we enhance its interpretation Consider that the last row (generally last n-1 rows) of T is some partitioning hyperplane as defined by

$$A_r = (1\ 0)^t$$

This hyperplane partitions the index space of the figure A-1 into bands The timing hyperplanes sweep all the index points in each band before another band is processed The band boundary must satisfy the equation

$$A_r * i_k \bmod 4 = 0 \qquad (A-2)$$

where i_k is an index point of the DG

Thus all index points inside one band are processed before another band is considered Inside each band the index points are swept by a family of parallel time hyperplanes $s = (1\ 1)^t$ Condition (A-2) assure that at any given moment no more than P index points are processed For each index point it is needed to determine *to which band it belongs and in what processor* it is mapped In order to distinguish between different bands, a coordinate b has been assigned to each band Thus an index point i_k will be assigned to a band

$$b = \left\lfloor \frac{A_r * i_k}{4} \right\rfloor$$

The allocation of computations to processors is done according to the formula

$$\hat{i}_k = A_r * i_k \bmod 4$$

For example the index point (5,4) will be allocated to processor (1) in band B_1

Further the exact processing time for each index point i_k depends upon the order in which the bands are executed Many execution orderings are possible

In table A-1, the original index set for the algorithm of figure A-1 and the allocation of indices to bands and to processors are shown Note that the execution of band B_0 followed by the band B_1 is chosen for the algorithm under consideration Thus the time coordinate in table A-1 was determined by first executing all points inside one band according to the ordering imposed by s and then moving to the next band Based on the information furnished in the table A-1 the architecture of figure A-3 is constructed as illustrated in chapters 3 and 4 Note that the dependencies are mapped by the transformation such that only local communications are required inside each band The communications between the computations in adjacent bands are performed via external FIFO queue registers

Note that the architecture of figure A-2 needs eight cells and results in an execution time of 22 clock cycles where as the architecture of figure A-3 needs only four cells and a FIFO of six memory locations and results in an execution time of 22 clock cycles (excluding the time to load the variables x and c) Note also how the computations are indexed (starting from 0 rather than 1) in order to have minimum number of bands and hence minimum execution time [10]

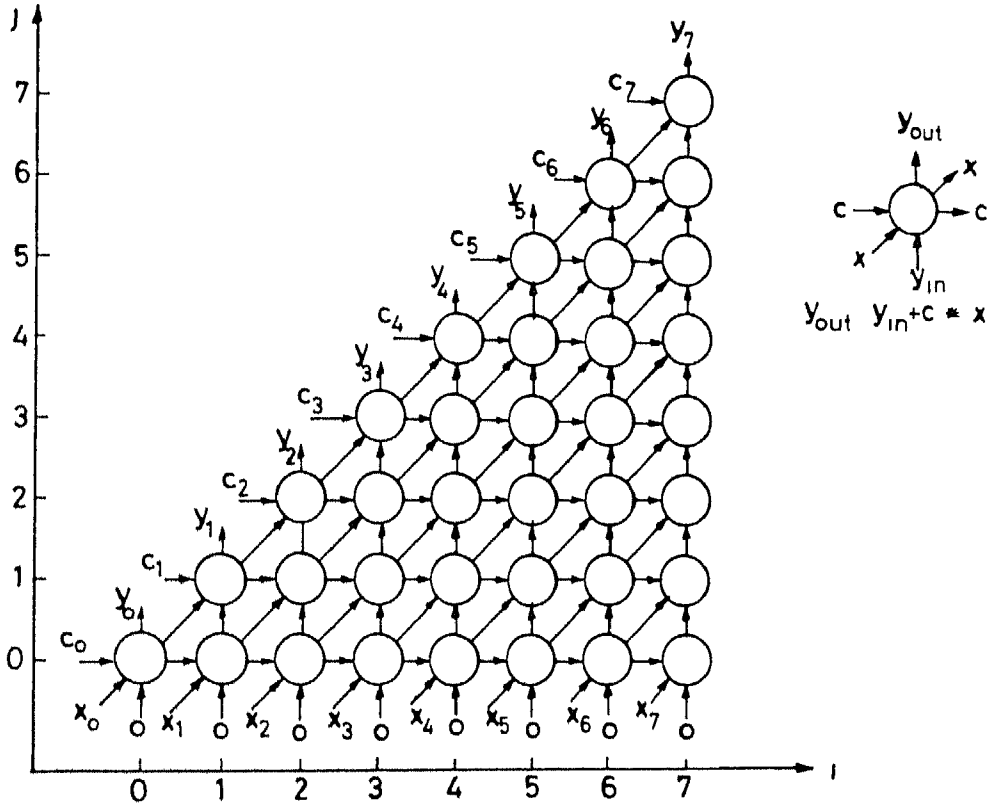


Fig A 1 DG for FIR filter

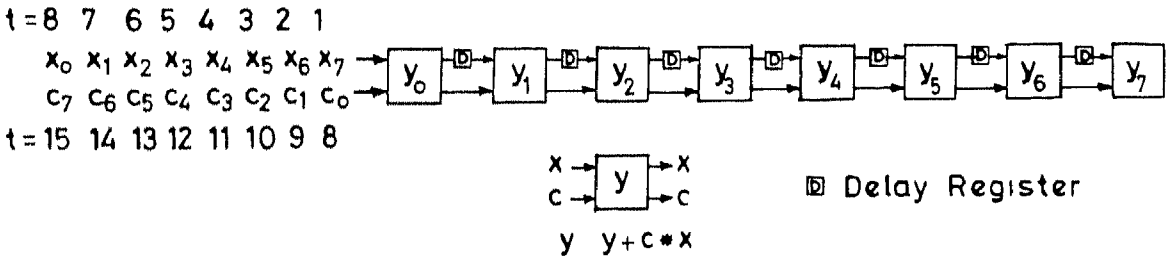
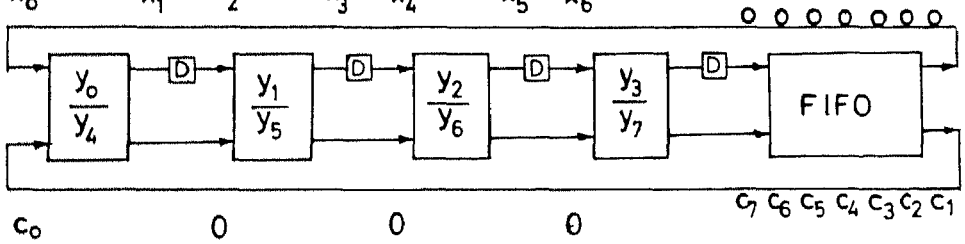


Fig A 2 Systolic solution for FIR filter, $p_-(0\ 1)^t$

Table A-1 Partitioned index set of FIR filter

band number	original index point		transformed index point (partitioned)	
	0	0	0	0
	0	1	1	1
	0	1	2	1
	0	2	0	2
B_0	0	2	1	2
	0	3	0	3
	0	2	2	2
	0	3	1	3
	0	3	2	3
	0	3	3	3
	1	4	0	0
	1	4	1	0
	1	5	0	1
	1	4	2	0
	1	5	1	1
	1	6	0	2
	1	4	3	0
	1	5	2	1
	1	6	1	2
	1	7	0	3
	1	4	4	0
	1	5	3	1
B_1	1	6	2	2
	1	7	1	3
	1	5	4	1
	1	6	3	2
	1	7	2	3
	1	5	5	1
	1	6	4	2
	1	7	3	3
	1	6	5	2
	1	7	4	3
	1	6	6	2
	1	7	5	3
	1	7	6	3
	1	7	7	3

t 22							x_0
t 21						x_0	x_1
t 20					x_0	x_1	x_2
t 19				x_0	x_1	x_2	x_3
t 18			x_0	x_1	x_2	x_3	x_4
t 17		x_0	x_1	x_2	x_3	x_4	x_5
t=16	x_0	x_1	x_2	x_3	x_4	x_5	x_6
t=15	x_1	x_2	x_3	x_4	x_5	x_6	x_7
t 14	x_2	x_3	x_4	x_5	x_6	x_7	0
t-13	x_3	x_4	x_5	x_6	x_7	0	0
t 12	x_4	x_5	x_6	x_7	0	0	0
t 11	x_5	x_6	x_7	0	0	0	0
t 10	x_6	x_7	0	0	0	0	0
t 9	x_7	0	0	0	0	0	0
t-8	0	0	0	0	0	0	0
t=7	0	0	0	0	0	0	x_0
t 6	0	0	0	0	0	x_0	x_1
t=5	0	0	0	0	x_0	x_1	x_2
t-4	0	0	0	x_0	x_1	x_2	x_3
t-3	0	0	x_0	x_1	x_2	x_3	x_4
t 2	0	x_0	x_1	x_2	x_3	x_4	x_5
t 1	x_0	x_1	x_2	x_3	x_4	x_5	x_6



t=1	c_0	0	0	0
t 2	c_1	c_0	0	0
t-3	c_2	c_1	c_0	0
t-4	c_3	c_2	c_1	c_0
t-5	c_4	c_3	c_2	c_1
t-6	c_5	c_4	c_3	c_2
t=7	c_6	c_5	c_4	c_3
t=8	c_7	c_6	c_5	c_4
t=9	0	c_7	c_6	c_5
t 10	0	0	c_7	c_6
t 11	0	0	0	c_7
t-12	c_0	0	0	0
t=13	c_1	c_0	0	0
t=14	c_2	c_1	c_0	0
t=15	c_3	c_2	c_1	c_0
t=16	c_4	c_3	c_2	c_1
t=17		c_4	c_3	c_2
t=18		c_5	c_4	c_3
t-19			c_5	c_4
t-20			c_6	c_5
t-11				c_6
t-22				c_7

Fig A 3 Partitioned systolic solution for FIR filter, p (0 1)^t

A 2 MULTIPROJECTION

In chapter 3 a methodology has been proposed which maps an n-dimensional DG onto an (n-1)-dimensional systolic array In principle, the method can be applied k times and thus reduces the dimension of the array to n-k This scheme is called the *'multiprojection method*

The potential difficulties of this mapping are (1) the presence of delay edges in the (n-1)- dimensional array and (2) the possibilities of loops or cycles, although no loops or cycles can exist in a DG

To handle the *cycle* problem, an *instance graph* (at certain time $t=t_0$) is defined as the array with all *self loops* removed Hence the instance graph has *no* loops or cycles According to the schedule s, all the nodes in the instance graph are executed simultaneously at $t=t_0$ An *activity instance* R_t (at $t=t_0$) can be defined as the nodes represented by the instance graph at $t=t_0$ Further according to the schedule s, there will be no overlap in time between two consecutive activity instances (one at $t=t_0$ and the next instance at $t=t_0 + D$)

Recall that a mapping methodology should consist of a schedule part and a projection part As to the schedule part it is always possible to find a valid schedule vector s' for the instance graph, by using the eqs (3-5) and (3-6), since there

is no loop or cycle in the instance graph And a valid projection vector p′ for the instance graph can be found according to the procedure outlined in chapter 3

To project an (n-1)-dimensional instance graph to an (n-2)-dimensional graph it is necessary to create a new type of delay denoted by τ Note that the global delay D and local delay τ are intimately related The relationship depends upon the constraints imposed by the *processor availability* To ensure processor availability an activity instance must have adequate time to be complete before the next activity instance starts So the following condition is necessary

$$D \geq \tau + (M-1)*(s' \quad p')*\tau \qquad (A-3)$$

where M is the maximal number of nodes along the p′ direction in the instance graph Note that this condition also guarantees that there is no time overlap between two activity instances

The *new transformation matrix* $T' = \begin{bmatrix} s't \\ A't \end{bmatrix}$ is formed where A′ is the processor basis corresponding to the new projection vector p′ Then *new transformed index set* and *dependencies* are computed (consider the *self loops* also) by considering the index set (last n-1 rows of the *old* transformed index set) and dependencies (last n-1 rows of the *old* transformed dependencies) of *all instance graphs*

Further, the new transformed index set has to modified by adding $(R_t-1)*[1+(M-1)*(s' p')]$ to the first row of the new transformed index set Thus the first row of the *modified new*

transformed index set specifies the *timestep* at which a computation will occur Since the *original delay link d with delay weight βD map to a link bearing delay weight βD+(s'd)τ* the new transformed dependencies are also modified by adding the first row elements of the *old* transformed dependencies multiplied by (D/τ) to the *corresponding* first row elements of the *new* transformed dependencies Thus the first row of the *modified new transformed dependencies* specifies the amount of *external delay* on various links With this information, the architecture is constructed as explained in chapter 3

For example consider the architecture of figure 4-4 which is obtained by projecting the 3-dimensional DG of matrix-matrix multiplication C=A*B algorithm in $(1\ 0\ 0)^t$ direction Note the original index set of this algorithm which represents the various computations Further from the transformed index set it can be found that there exists seven activity instances viz at t=3,4, ,9 By considering the instance graph at t=6 a valid schedule vector $(1\ 0)^t$ is found which results in minimum computation time Further there exists three valid projection vectors namely $(1\ 0)^t$, $(1\ 1)^t$ and $(1\ -1)^t$ Consider the projection vector $(1\ 1)^t$ for further discussion

Then from eq (A-3), D≥3τ because M=3 Let D=3τ The geometric transformation matrix, corresponding to this projection vector is $T' = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$ The *new transformed index set* and *dependencies* are given here

$$T'*I^2 = \begin{bmatrix} 2 & 3 & 4 & 3 & 4 & 5 & 4 & 5 & 6 & 2 & 3 & 4 & 3 & 4 & 4 & 5 & 6 & 2 & 3 & 4 & 3 & 4 & 5 & 4 & 5 & 6 \\ 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \end{bmatrix}$$

$$T'*D = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The *modified new transformed index set and dependencies* are printed below

$$T'*I^2 = \begin{bmatrix} 4 & 10 & 6 & 10 & 14 & 10 & 14 & 18 & 8 & 12 & 16 & 12 & 16 & 20 & 16 & 20 & 24 \\ 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \end{bmatrix}$$

$$T'*D = \begin{bmatrix} 4 & 3 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

From $T'*D$ it can be noted that the variables a and b stay within the cells and they need an external delay of three and two clock cycles respectively This means that a particular element of the matrix A will be used again and again after every three clock cycles and that of matrix B will be used after every two clock cycles Following the guidelines provided in chapter 3 for constructing the systolic array from the transformed index set and dependencies, the architecture of figure A-4 is obtained from $T'*I^2$ and $T'*D$

Note that the architecture of figure 4-4 needs nine cells and the total required execution time of seven clock cycles (excluding the time needed to load the elements of matrix B) In the architecture of figure A-4 there are only three cells but the total required execution time is 23 clock cycles (excluding the time needed to load the elements of both matrices A and B) Further the cells of the architecture of figure A-4 needs some *local* memory to store the elements of matrices A and B

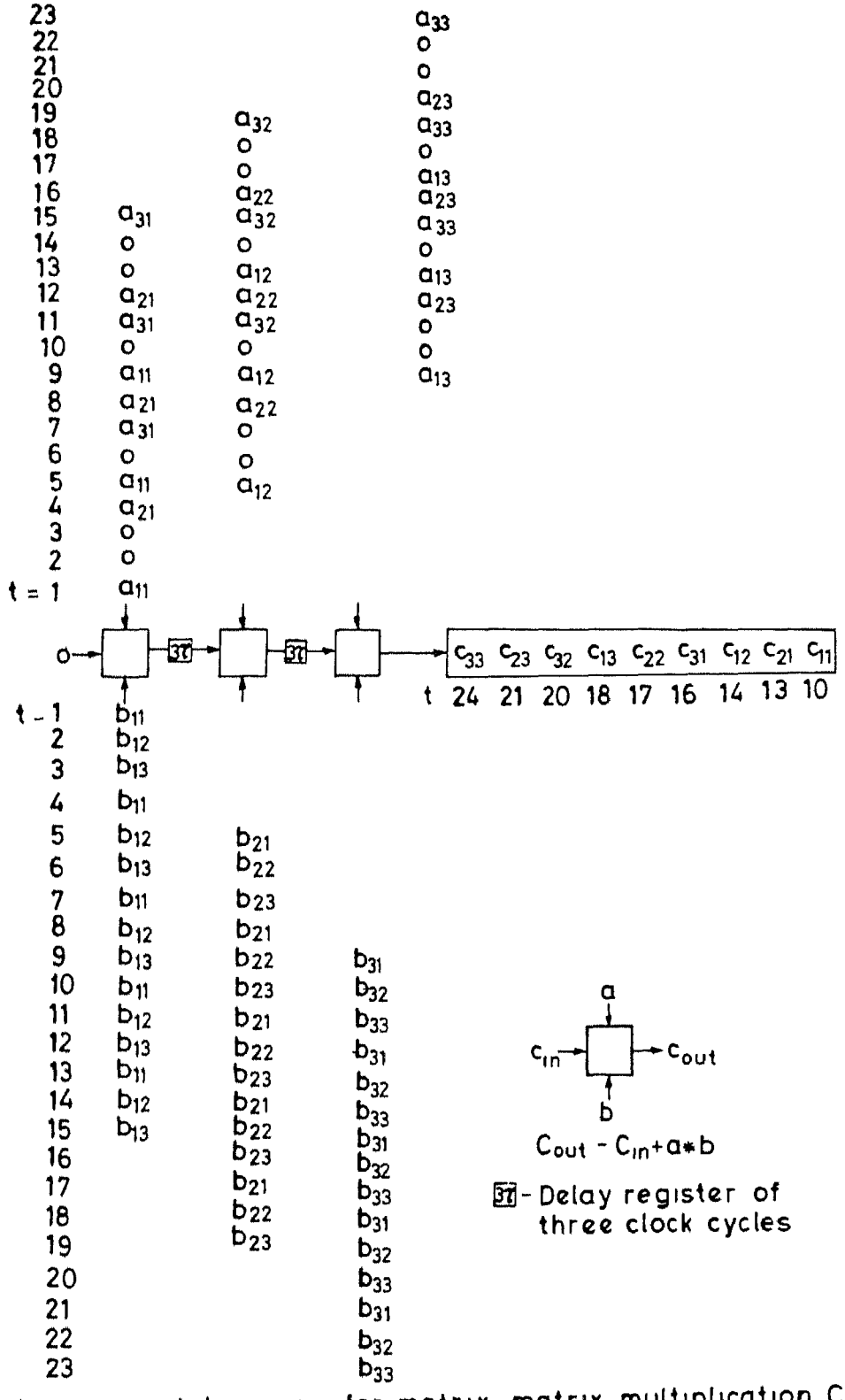


Fig A 4 Linear systolic array for matrix matrix multiplication C A*B, p $(100)^t$, p'-$(11)^t$

LIST OF REFERENCES

[1] H T Kung 'Why Systolic Architectures? IEEE computer, 15(1) Jan 1982

[2] S Y Kung On Supercomputing with Systolic/Wavefront Array Processors' Proc IEEE Vol 72 No 7 1984 pp 868-884

[3] M Marchesi G Orlandi and F Piazza Parallel Architectures for Transforms in Digital Signal Processing Signal Processing-IV Theories and Applications, edited by J L Lacoume et al Elsevier Science Publishers B V (North-Holland) pp 775-778 (1988)

[4] C Mead and L Conway 'Introduction to VLSI systems Addison-Wesley, Section 8 3, 1980

[5] Earl E Swartzlander Systolic Signal Processing Systems Marcel Dekker Inc 1987

[6] G J Li and B W Wah, The Design of Optimal Systolic Arrays",IEEE Trans on Computers Vol C-34 No 1 Jan1985

[7] C E Leiserson, F M Rose and J B Saxe Optimizing Synchronous Circuitry by Retiming', Third Caltech Conf on VLSI, edited by R Bryant, Comp Sci Press 1983

[8] W L Miranker and A Winkler 'Space-Time Representations of Computational Structures", Computing, Vol 32 pp 93-114 1984

[9] P Quinton, Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations Proc 11th Ann Symp on Comp Architecture 1984

[10] D I Moldovan and J A B Fortes Partitioning and Mapping Algorithms onto Fixed Size Systolic Arrays , IEEE Trans on Computers Vol C-35 No 1, Jan1986

[11] D I Moldvan, On the Design of Algorithms for VLSI Systolic arrays , Proc IEEE Vol 71, No 1 Jan 1983 pp 113-120

[12] ---------- 'ADVIS A Software Package for the Design of Systolic Arrays , IEEE Trans on Computer Aided Design Vol CAD-6, Jan 1987 pp 33-40

[13] N Faroughi and M A Shanblatt, 'Systematic Generation and Enumeration of Systolic Arrays from Algorithms' 1987 Intl Conf on Parallel Processing, 1987 pp 844-847

[14] ---------- "An Improved Systematic Method for Constructing Systolic Arrays from Algorithms' 24th ACM/IEEE Design Automation Conference Miami Beach, Florida June 1987, pp-26-34

[15] ---------- "A New Method for Construction of Systolic Arrays", Ph D Dissertation, Deptt of Electrical Engineering, Michigan State University, 1987

[16] S Y Kung S C Lo, S N Jean and J N Hwang, Wavefront Array Processors - Concept to Implementation', Computer July 1987 pp 18-33

[17] D S Broomhead, J G Harp, J G Mcwhirter, K J Palmer and J G B Roberts "A Pratical Comparison of the Systolic and Wavefront Array Proceesing Approach' 1985 ICASSP, March 1985, pp 297-299

[18] S Y Kung, "VLSI Array Processors', Prentice Hall, Englewood Cliffs, 1988

[19] J A B Fortes, K S Fu and B W Wah Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays , 1985 ICASSP, March 1985, pp 300-303

[20] D E Heller "Partitioning Big Matrices for Small Systolic Arrays', VLSI and Modern Signal Processing, edited by S Y Kung et al, Prentice-Hall, Englewood Cliffs pp 185-199 (1985)